Issue 14 | Jan 2008

blender art magazine

Blender learning made easy

Cartoon & GE

COVERART - Cristian Mihaescu 'Hope for a dinner'

EDITOR
Gaurav Nawani gaurav@blenderart.org

MANAGING EDITOR
Sandra Gilbert sandra@blenderart.org

WEBSITE
Nam Pham nam@blenderart.org

DESIGNER
Gaurav, Sandra, Alex

PROOFER
Kevin Braun
Phillip Ryals
Bruce Westfall
Joshua Leung
Lynda Schemansky
Eric Pranausk
Noah Summers
Joshua Scotton
Mark Warren
Wade Bick
Patrick O'Donnell
Brian C. Treacy
Scott Hill
Henriel Veldtmann

WRITERS
Brian Cordell Hynds
Brian Treacy
Husam Ibrahim
Igor Krianovskij
ititrx
John Buresh
Mal Duffin
Nanmo
Rogério Perdiz
Tobias Dahl Nielsen

COVER ART
Cristian Mihaescu 'Hope for a dinner'

CONTENTS

EDITORIAL

Sandra Gilbert
Managing Editor

Even after years of using Blender, the wide scope of possibilities for creativity in Blender still amazes me. You can create whatever you imagine, from cartoons to games. And in this issue, it is that very diversity that we are going to explore.

Growing up I was always fascinated by cartoons, and to this day I still enjoy watching cartoons. Especially with all the advancements in technology paving the way to ever more amazing feats of animation possibilities.

3D modeling programs have changed the look of cartoons and yet the basic concepts developed by the “cartoon masters" remain the same. 3D programs, including our beloved Blender, allow us the freedom to create cartoons in any style that appeals to us. Something that, while possible, was far more difficult using traditional "cel" techniques. From the "traditional" look to anime, cartoons are becoming a popular medium of artistic expression.

As 3D animation becomes more popular, the line between cartoon and animation converges and continues to blur, causing the two words to become interchangeable. In fact with the growing number of full length 3D animation movies being produced, the definition of cartoon is expanding as fast as the imaginative minds of the animators creating them. Decreasing costs of higher priced modeling software and free programs like Blender let anyone try their hand at creating their own cartoons and animations, opening up the cartoon/animation field to a new wave of young cartoon lovers and would be creators.

Building off the growing popularity of cartoons, games are quickly becoming an artistic force in their own right. These days, you can find games based on many popular cartoons, animations and movies as well as original concepts. In fact “gaming” has become a big industry with growing opportunities for those skilled in modeling, texturing and yes even character animation. Many game companies are seriously in need of 3D artists, and I am betting that need will continue to grow as technology advances and the possibility for more graphic intense games grows. Game creation uses a lot of the same skills and talents as animation. Knowledge of modeling, texturing and character animation is a great help whether you are creating your first game masterpiece or your hundredth.

Of special interest to us is the fact that Blender can be used to either create your game assets for exporting to a separate game engine or you can create games right within Blender itself. The Blender Game Engine has seen a great number of improvements over the years, leading to a growing segment of our community learning to develop game assets, demos and full games. As the community grows, solutions are being found, new techniques are being created and ways to use Blender's logic bricks more effectively both with and without additional Python coding has led to a growing number of games being created and released in the GE forums. The variety of games being created is as imaginative as the artists that have devoted their time to the GE.

So whether you are an animator, budding game creator or simply curious about areas of Blender that you haven't yet learned, we have you covered. We have gathered up some really great articles to get your creativity up and running.

Happy Blending!
sandra@blenderart.org

Good composition can make a boring image stand out and make you sit up and say 'Wow'!

After recently seeing Virgilio's blog post on composition and framing, I realized that although I have known about the "Rule of Thirds" for some time, that it was no longer something I really thought about. Which is a shame, because good composition can make a boring image stand out and make you sit up and say 'Wow'!

To a certain extent, a lot of artists use the 'Rule of Thirds' subconsciously when they create their images, which oddly enough isn't as strange as you would think. There are certain proportions and placing of focus points that humans just instinctively respond to. So quite a few of us just place objects instinctively and move them around until it 'looks' right.

But like so many things in life, taking a moment to consider the impact of composition for your image or animation shots, can create a more dynamic image that fully engages the viewer.

Realizing that composition was no longer a conscious decision when I sat down to create images, I went back and looked at some of the images I have done over the years. Not surprisingly, images that I was really pleased with followed the 'Rule of Thirds' and images that never quite seemed to work suffered from bad composition and a total lack of focal point.

I found the exercise extremely enlightening and encourage anyone seeking to improve their images and animations to try it for themselves. When you are done looking at your own images, try it with other images, remembering to look a variety of images, both good and bad.

If you are not familiar with the 'Rule of Thirds', I suggest you check out the following two articles. They both have wonderful explanations and visual examples of excellent compositional practices.

Learning Composition with the New BlueSky Trailer

www.methart.com/tutorials/composition.html

Game Engine Resources, Games and Demos

The Game Engine forum has a number of excellent tutorials and demos explaining various parts of the Game Engine, as well as methods to set up a number of different types of games assets such as menus, Computer/enemy AI, shooting projectiles, etc. The forums contain a wealth of information that would be far too lengthy to list here. So I am going to mention just a few that I have seen recently that have caught my eye as extremely useful and resourceful uses of GE abilities.

- Level Editor Template
- Autonomous bots without python
- Simple AI
- 3rd Person Adventure Game Template
- Ray sensor shooting tutorial
- BGE FPS Template

Official Documentation can be found on the

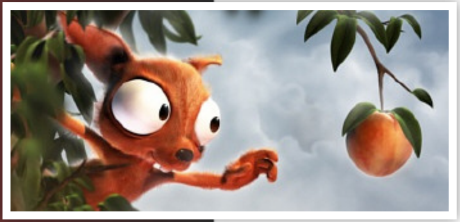
- Blender Wiki :
- Game Engine Documentation
- BSOD Introduction to the Game Engine

Good composition can make a boring image stand out and make you sit up and say 'Wow'!

Sampling of Games and Demos currently in production

More information and updated progress on these games/demos can be found by clicking the listed links.

- 1st MBGP - Kart Racer! : actually a new learning project that will last one month, at which point a new project will be started.
- Project Echo
- My First RPG Game (by Rusty246)
- Duk! Duk!
- GunFlash
- My Tetris Game
- Shoot'n'score



Project Peach

The Peach Project is merrily trucking along, bringing exciting improvements and features to Blender. My favorite, of course, is the new hair/fur capabilities. If you haven't yet seen the results of the new hair and fur, you are surely missing out.

In order to use the same cool new features as the Peach team, you will need to check out the development SVN repository, since the needed features have all been added since 2.45

New Features Currently in SVN

You can already find some projects in the trunk of SVN. Testing and bug reporting is encouraged. You can read more about them here.

- Skinning Improvements
- Rendering Features
- Mesh Deform Modifier
- Pole Target for IK
- Glossy Reflection/Refraction
- Raytraced Soft Shadows
- QMC Sampling
- Weight paint visualization and Multi Modifier
- Render Baking and Normal Mapping
- Particles and Strands

Project Apricot

Project Apricot team members will meet 2nd week of January for a long weekend to prepare for production issues, but the real start will be the first week of February. Only then will real decisions be made on game concept, game design, and other targets. However, we do know it'll be derived from Project Peach's furry & crazy characters in a forest.

As for the previous open movies, a pre-sale campaign to purchase the DVD set in advance will be organized to allow the Blender and Crystal Space communities to support this project. Again, there's the ambitious target of reaching a 1000 DVDs sold in advance, before February 1st.

Expect on the DVD the complete game, all content in Creative Commons Attribute, video blog material, clear examples and/or tutorials for how to extend the game, add levels, change characters, and so on. Early purchasers of the Apricot game will get their names credited on the game intro and final credit roll.

And don't forget: although we aim at a creating compelling gaming content, the real Apricot target is to improve the open source pipeline for professional game creation. This focused around Blender as a modeling and animation tool, Crystal Space as 3D engine and delivery platform, and some Python scripting magic to tie it all together.

Thanks for the support!

-Ton-

Precision Modeling PDF Guide

You may remember Robert Burke from the excellent tutorial (Producing Models With Accuracy) he wrote in BlenderArt Magazine's Issue #11 (Mechanical). We are happy to announce and promote his expanded work with precision modeling in blender. It looks to be an excellent reference that we encourage all blender users to check out.

Description from Robert Burke's website

The Precision Modeling Guide to Blender is a 151 page walk-through of Blender's mesh modeling tools and design techniques. Illustrated with over 600 images, it gives a clear and concise introduction to using Blender. By progressing through the guide you will be introduced to many of Blender's commonly used modeling tools and also a few of the not so well known ones. The tools are presented in a real working design example with explanations of why the tools are used and not merely a demonstration of the tools in use.

Don't be fooled by the name, the guide is useful to anyone wanting a clear introduction to Blenders mesh modeling tools.

Stats: 151 Pages ; 613 Images; 4.85 Mb

Download the Precision Modeling Guide here.

3D WORKSHOP: Modeling and Rigging a Cartoon-looking Spider

Toony Spider

by Igor Krianovskij

Introduction

Blender's modeling tools are excellent for many tasks, especially for making organic forms. To make a funny, simple cartoon looking character with Blender is as simple as a walk in the park (well, once you know how, obviously). I'll try to prove that in this short tutorial. So, first you'll need an idea. If you don't have a fresh idea, use an old one, it's probably already mature enough to be recycled with some fresh dimensions. Such was the case for me... Actually, you could even skip this part; with Blender you don't even need an idea. To play with the tools and their options is usually more than enough to start one!

Level: Beginner to Intermediate

My old spider illustration was a perfect starting point for a short and silly 3D movie, and to check out some new Blender tools in areas such as: lights, scenes, painting, materials, rigs, particles, shape keys, NLA, nodes, video compositing, sound, codecs, etc...

So I needed a good looking model, but one that is also simple enough to achieve decent/short rendering times. You know, short rendering times = longer movie in a shorter time.

There are many good modeling tutorials around, including ones in previous BlenderArt volumes (http://www.blenderart.org/), so check them out. Most of the modeling tools are already mature, so they haven't changed much.

Here I'll show you my key stages of the modeling and rigging process:

by Igor Krianovskij

MODELING TIP

The basic modeling tools are: Grab & move [G], Scale[S] and Rotate [R]. You can select joined mesh pieces with [L] in Edit Mode (modeling is done in this mode most of the time). Once selected, you can separate them again with [P]. To make a face (also known as a polygon) select edges or vertices and press [F]. With [ALT+S] you can move faces along their normals.

BODY

It's good to start your model in the center of the 3D space, especially if you'll use Mirror Modifiers, Armatures, etc. So first hit [SHIFT+C] to move the 3D cursor to the center, and "NumPad1" for the Front view (so that the "X" axis goes from left to right).

Add a cube (Object mode: [SPACE], Add › Mesh › Cube) and add a Subsurf Modifier to it ([F9] for the Editing buttons, then Add Modifier › Subsurf), set the Level to 2 and hit the Apply button (image 1.h) (or [ALT+C] in object mode) in the Modifiers panel. Tweak the vertices to get a shape like this.

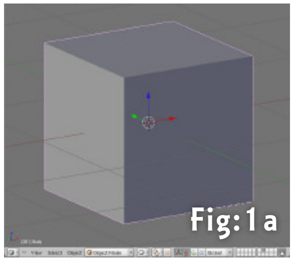


Fig:1a

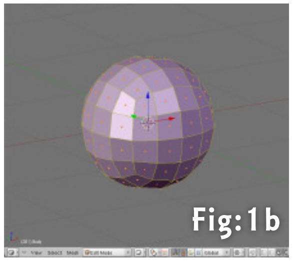


Fig:1b

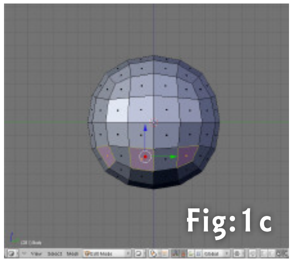


Fig:1c

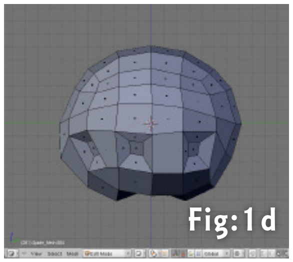


Fig:1d

Select three faces and extrude them inward with [E] to make a place for the upper legs 'connections' with the body. You can delete one side of the body mesh and add a Mirror Modifier, so you'll need to tweak only one half of the body since it's usually good to build things symmetrically at first. OK, this is the spider's body shape. Easy, huh!? We'll add hair or fur later...

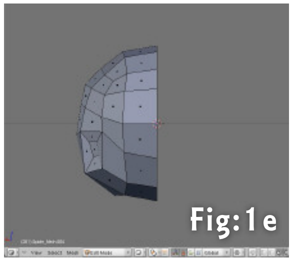


Fig:1e

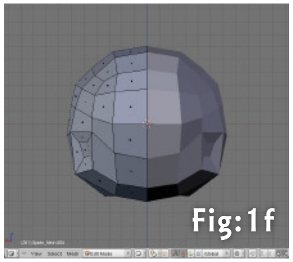


Fig:1f

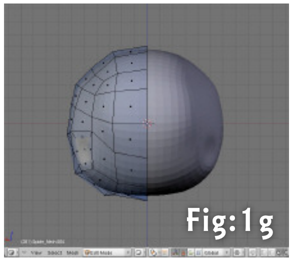


Fig:1g

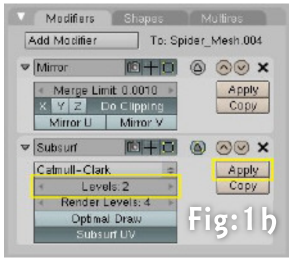

Modifiers | Shapes | Multires
Add Modifier | To: Spider_Mesh.004
Mirror: Merge Limit 0.0010; X Y Z; Do Clipping; Mirror U; Mirror V; Apply; Copy
Subsurf: Catmull-Clark; Levels: 2; Render Levels: 4; Optimal Draw; Subsurf UV; Apply; Copy

Fig:1h

by Igor Krianovskij

HEAD

Check the sketch to see how big the head must be to fit the style of the illustration. You can be flexible here, making it bigger or smaller. Sometimes you get better results when you don't follow the sketch exactly.

There are many ways to build the head for a body. I used the topology of the body as the starting point for the head. So, I duplicated body faces (4x3) in Edit Mode with [SHIFT+D]. These 12 new faces are still part of the body mesh, so I separated them using [P].

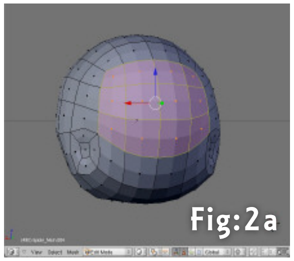


Fig:2a

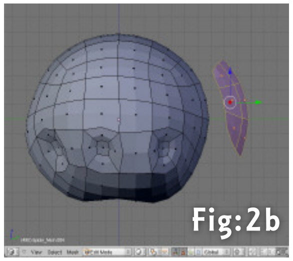


Fig:2b

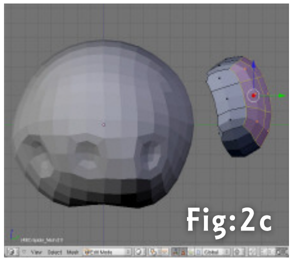


Fig:2c

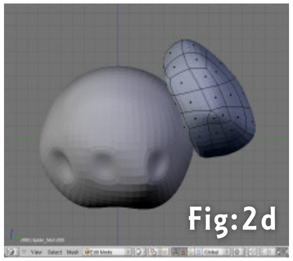


Fig:2d

You can make the head in the same way as the body above... from a cube. But by duplicating part of the already existing body mesh you get precise alignment of vertices which are useful for quicker assembly of body parts (less tweaking, faster result!).

So, after a few extrusions [E] and some tweaking of some vertices, I came up with this:

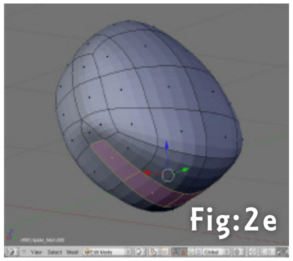


Fig:2e

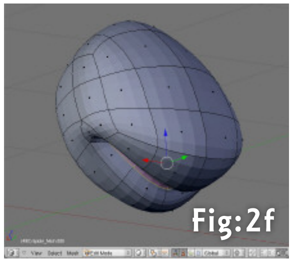


Fig:2f

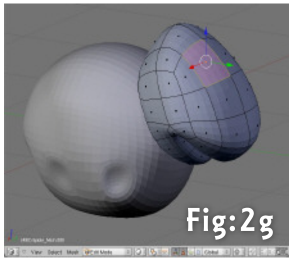


Fig:2g

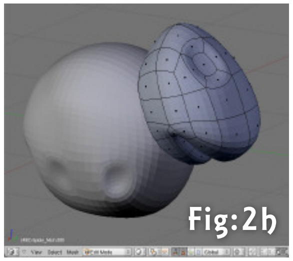


Fig:2h

by Igor Krianovskij

MODELING TIP

In Blender, [X] is an extremely important HotKey, which is usually used to delete ugly & awkward looking artwork! Interesting how efficient Blender is with a few HotKeys, huh!?

LEGS

Again, with the same principle as a head, I selected one (which was enough for this part) face where the middle leg "joint" will be located and duplicated it with [SHIFT+D] and separated it with [P]. After a few extrusions [E] and vertex tweaks, I came up with this:

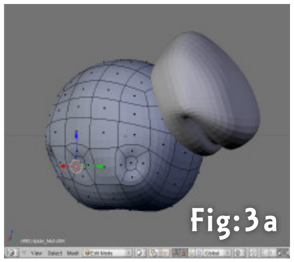


Fig:3a

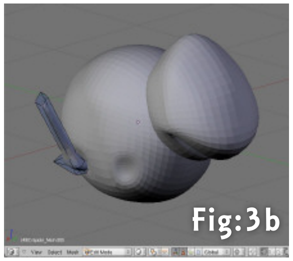


Fig:3b

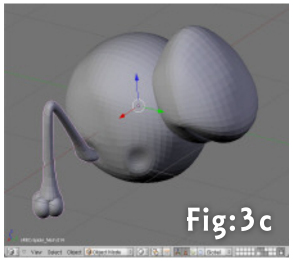


Fig:3c

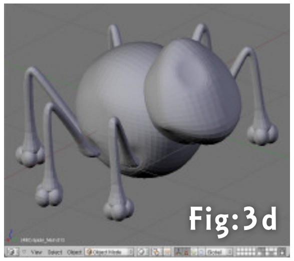


Fig:3d

At the tip of the leg I added four spheres (see footnote 1) and joined all leg meshes together with [CTRL+J]. I could also pose the leg into a different rest position, but for a later stage it's important that the knees are at least a bit bent for the proper Armature IK calculations needed later. Finally I duplicated the remaining five legs in the planned positions (again, with [SHIFT+D]).

EYES & TEETH

The spider's teeth are super simple... well like the rest, huh!? OK, I admit, I went about this the same way one more time! [SHIFT+D], [P], [E], tweaking... But this time I added an extra Loop Cut with [K] (image 4c.) to make the topology of the spider's teeth more fascinating!

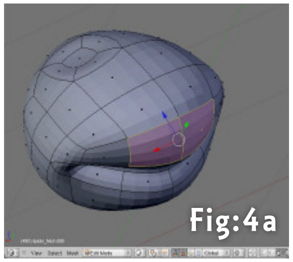


Fig:4a

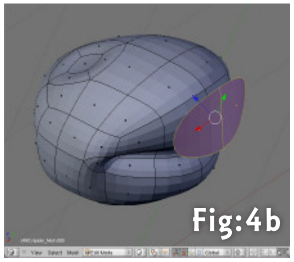


Fig:4b

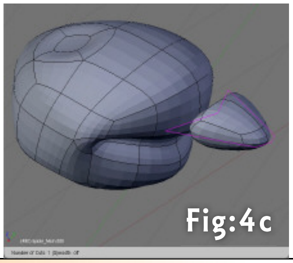


Fig:4c

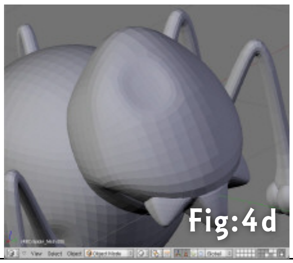


Fig:4d

MODELING TIP

Use Median Point and 3D Cursor pivots with the Snap tools [SHIFT+S] for the center positioning of an eyelid axis. Face Loops and Edge Loops can be selected with [ALT+SHIFT+RMB] and [ALT+CTRL+RMB]. Sneaky tools!

New Particle System: Check the BlenderWiki ActionBook for updated versions! http://wiki.blender.org/index.php/Blender_3D:_ActionBook

by Igor Krianovskij

I could make the eyes even simpler, but there's a little trick for an eye's 'iris lens'. Namely, you get better light reflections if you make the lens more spherical or convex. Here's how I made it: Add a sphere using [SPACE], then Add › Mesh › UVsphere (with 8 Segments and 8 Rings). Select the top 16 faces (2 rings) and separate them with [P]. Make the lens more convex using the Move [G], and Scale [S] tools.

The eyelid is made with an additional UV-Sphere (12 Segments and 8 Rings), rotated 90 degrees relative to the eyeball. Then scale, delete one segment row and extrude an Edge Loop toward the center of the eyelid (images 5 b-c.) For the opening/closing of the eyelid I used

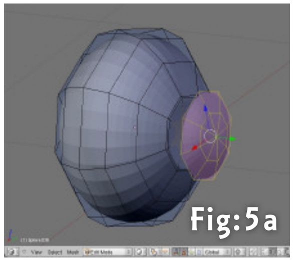


Fig:5a

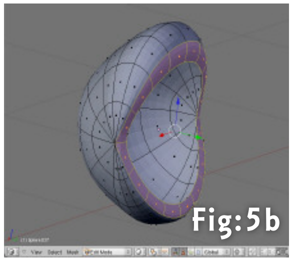


Fig:5b

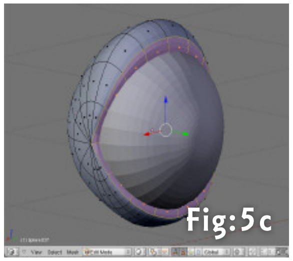


Fig:5c

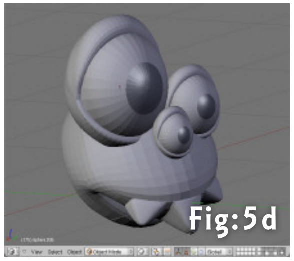


Fig:5d

ShapeKeys, but I won't go into ShapeKey details at this time. You can check the related file for details.

Hair , Fur & Other Horrendous Growth!

The spider model at this stage is basically done. The rest are colors and of course horrendous hair! That's quite easy again. Colors were tweaked in the Shading Menu [F5] and the Particle Emitters used duplicated parts of the already made spider meshes.

The Particle System in the current Blender development versions (Jan. 2008) is so advanced that this file made in Blender version 2.45 is outdated already. So, particles in this file will not be compatible in future Blender versions! Check the related file for these Particle Settings (use [F7] for the Object(Physics)buttons and Particles panel).

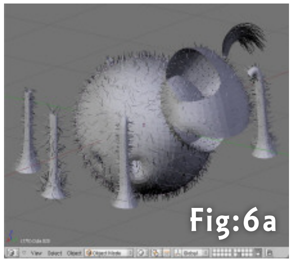


Fig:6a

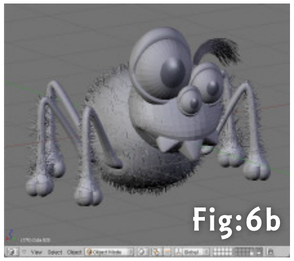


Fig:6b

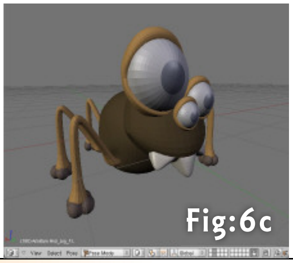


Fig:6c

Fig:6d

RIGGINS TIPS

If you want to avoid some unpredictable Armature behavior, start to build the rig in the center [SHIFT+C]!

If you want to save time in naming bones, name only 'important' bones (Root_*, Handle_*, Tag_*) and leave Blender to name the rest of the bones automagically! Only take care to add *.R or *.L on the end of certain bone names (for example: Handle_Leg_01.L). Blender will recognize this as a bone chain on the Left side. This is also useful for Mirror Bone editing!

by Igor Krianovskij

Rig, Oil & Sweat!

The next phase toward animation was rigging. This armature contains basic (version 1 - see footnote 2) use of IK constraints, but for an animation I didn't need a more complex rig. So, I added the first bone at the center (Object mode: [SPACE], then Add › Armature). I usually name this first bone Root_'something', in this case Root_Spider.

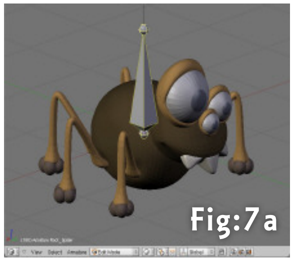


Fig:7a

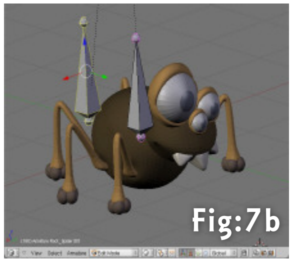


Fig:7b

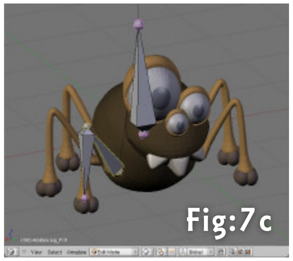


Fig:7c

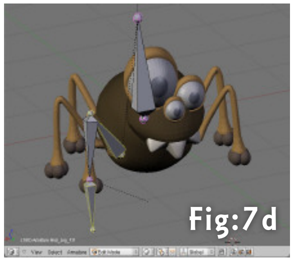


Fig:7d

Afterward, I duplicated this bone in Edit Mode, extruded [E] the tips of the bones two times and positioned them again with Snapping tools [SHIFT+S].

Here is an example of setting a leg bone in the exact position, which is very hard to achieve by hand, especially when working with organic forms and exotic angles:

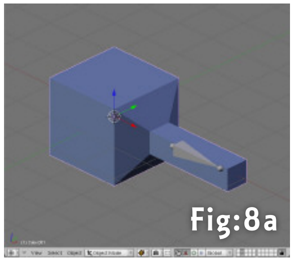


Fig:8a

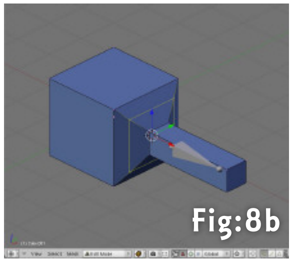


Fig:8b

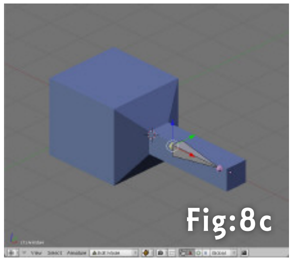


Fig:8c

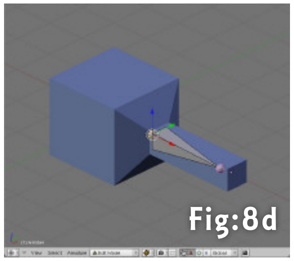


Fig:8d

8a) Add Bone ; **8b)** Set pivot to Median Point; select Edges; [SHIFT+S], then choose Cursor -> Selection ; **8c)** Select the tip of the bone in Edit Mode ; **8d)** [SHIFT+S], then choose Selection -> Cursor

RIGGINS TIPS

You can switch between Pose and Edit Modes with [TAB], and Pose and Object Modes with [CTRL+TAB].

Don't forget to disable Envelopes in the Armature Modifier if you're using Vertex Groups!

FINAL MASTER TIP: One animation per day keeps Kaito away!

by Igor Krianovskij

OK, back to the spider's rig. After I extruded and positioned all needed bones for a leg, I started with an IK chain logic setup. Bones with IK constraint solvers are yellow colored. Every spider's leg has one IK solver (set to ChainLen:2) which I added in a Pose Mode (image 9b.). After I duplicated the left/right side legs [SHIFT+D], I then added a bone for the head and one bone for a mouth/jaw. There is also an optional handle for moving all legs simultaneously!

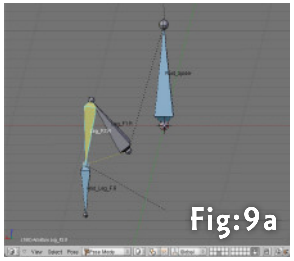
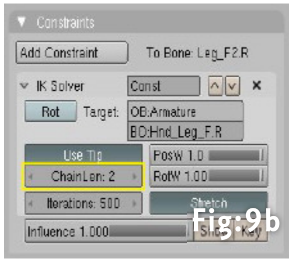


Fig:9a


Constraints

Add Constraint   To Bone: Leg_F2.R

IK Solver   Const

Rot   Target: OB:Armature
BO:Hnd_Leg_F.R

Use Tip   PosW 1.0
ChainLen: 2   RotW 1.00
Iterations: 500   Stretch
Influence 1.000   Show   Key

Fig:9b

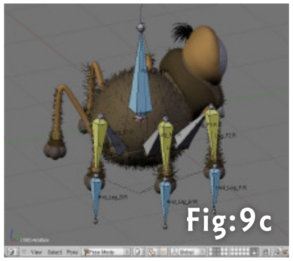


Fig:9c

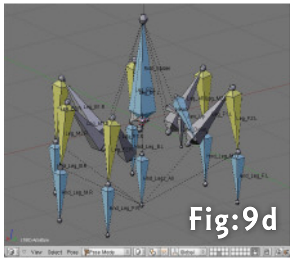


Fig:9d

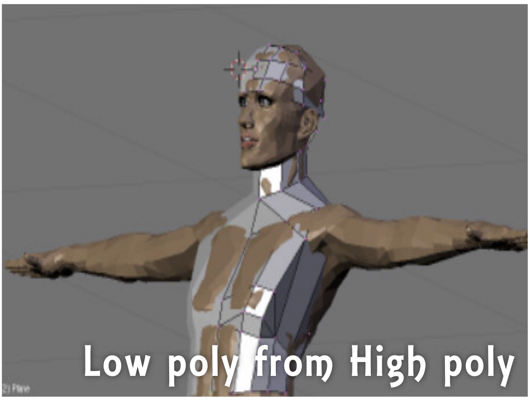
Finally all that was left to do was to attach the mesh to the Armature. That's also quite easy. I added an Armature Modifier to the mesh and assigned Vertex Groups to each bone.

Download the Spider.blend here:

Spider in the park. - Notes

1. Adding the spheres this way isn't optimal if you're building the model for a game. There must not be hidden faces in that case! This model was planned for an animation, so these 'dense' spheres can be used as particle emitters if needed.

2. The Armature system in Blender was build in stages. Version 1 can do FK, IK and some other basic constraints. Version 2 can do more advanced things (check Mancandy's rig made by Bassam Kurdali as an example). Version 3 is the latest and greatest Armature system used for characters in the Blender Institute's next "Open Movie"! ●



by ititrx

Introduction

This is a short tutorial explaining a technique to model a low poly Game Engine character over a high poly model. There are many techniques to accomplish this (box modeling, combing shapes, etc) so this is just one of them. Please use your favourite modeling techniques while following along.

Level: Beginner to Intermediate

Open Blender and delete the default camera and cube. Select layer 2 and import a high poly model of your choice. Divide your screen into three views, two 3D views on the top and the buttons view on the bottom. Choose one 3D view for the front view and one for the side view.

Select layer 1 and layer 2. You will make your model on layer 1 over the model on layer 2. You can make your planes as large as you care to, and as close to the model on layer 2 as you choose. When you want to check your model, just click layer 1. Another option would be to just leave the the model on layer one and hide/unhide it for use.

STEP1: Now place the 3D cursor on the forehead in the front view and check alignment in the side view as illustrated in Fig 1. From front view, insert a plane.

STEP2: With the plane still selected and in edit mode, add a mirror modifier, as in Fig 2, to simplify the modeling task.

STEP3: Now grab the plane and move it to the right on the X axis as in Fig3.

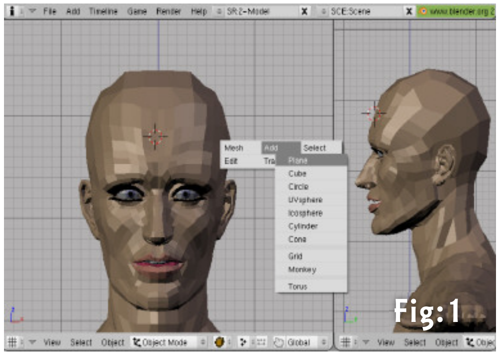


Fig:1

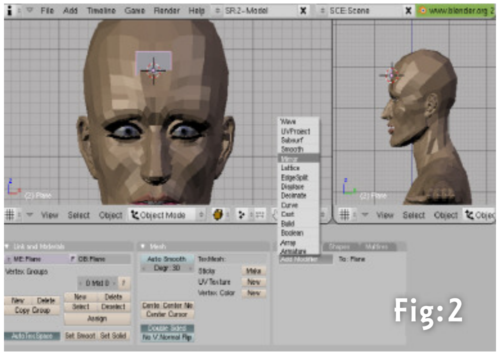


Fig:2

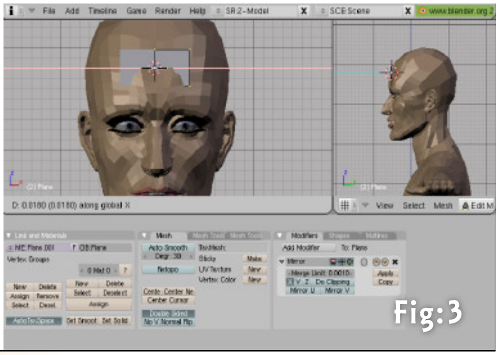


Fig:3

STEP4: From the side view, rotate the plane on the X axis to match the angle of the model, Fig.4.

STEP5: Select the right edge of the plane and extrude it on the X axis, Fig.5.

STEP6: With the right edge still selected, grab on the Y axis and pull it into the edge of the model on layer 2, Fig 6. See how the bottom vertex tips into the model below?

STEP7: Extrude that same edge again on the X axis, and then move the edge in towards the model, Fig. 7.

STEP8: Fig. 8 shows the results of extruding the two top edges of the two front faces up and toward the back until the planes touch the model underneath.

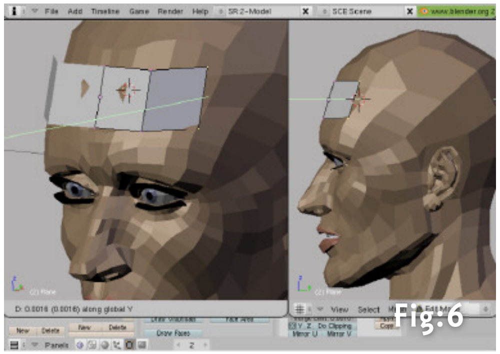


Fig:6

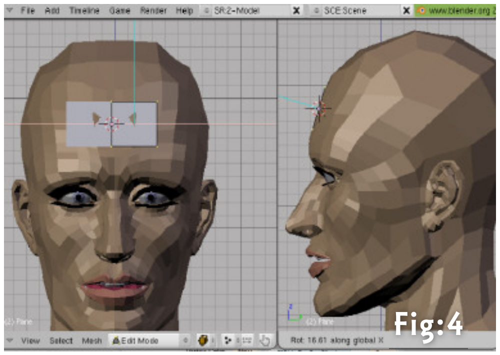


Fig:4

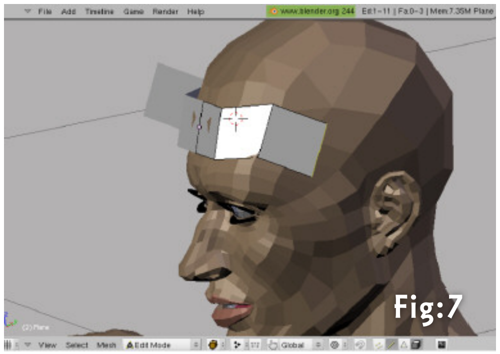


Fig:7

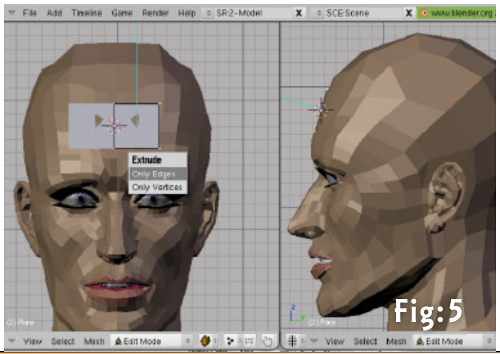


Fig:5

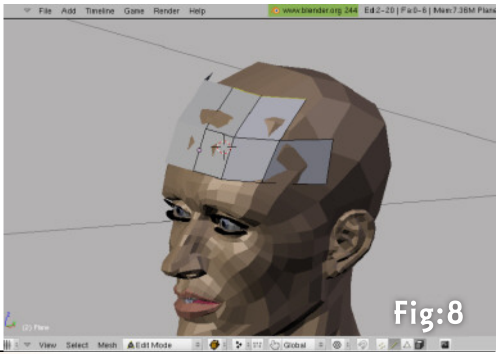


Fig:8

by ititrx

STEP9: Continue in this manner, covering the model's head on layer 2, Fig 9.

STEP10 Fig. 10 and Fig.11 illustrate the selection of layer 1 only to modify the existing planes. Two points were selected and merged into one.

STEP11: Fig. 12 and Fig. 13 illustrate face creation from the selected vertices to cover/close the gap between the faces.

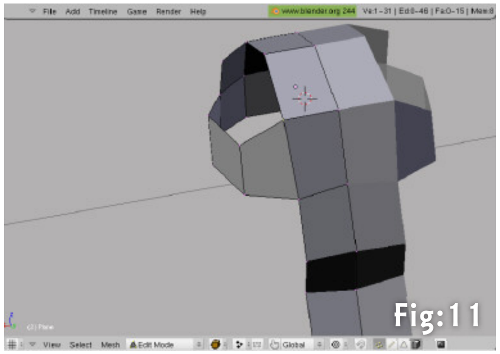


Fig:11

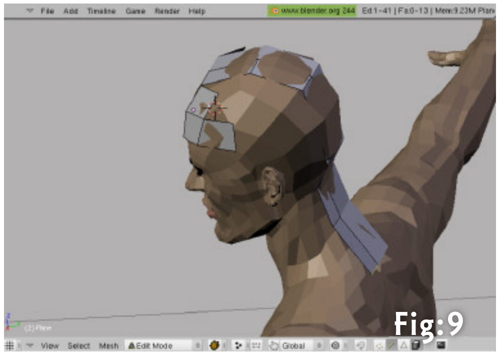


Fig:9

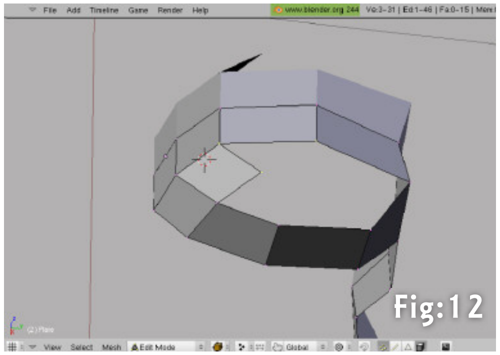


Fig:12

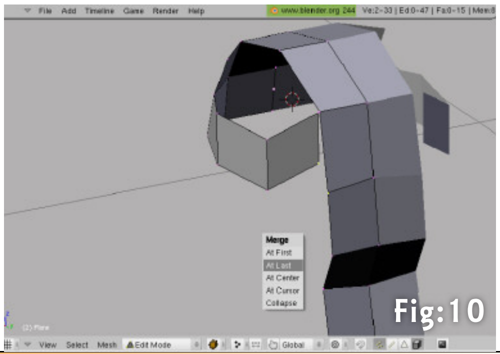


Fig:10

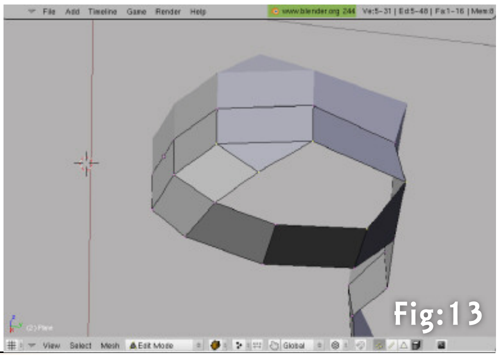


Fig:13

by ititrx

by ititrx

STEP12: Selecting layer 1 and 2 again, you can see there are some ugly 'point' where the vertices stand out too far from the model, Fig. 14.

STEP13: With some vertex tweaking, the horns are removed, Fig.15.

STEP14: In Fig. 16 the lower edges of the bottom faces were extruded.

Continue in this manner to cover the whole model. Making changes where necessary to cut the polys down. With practice you may be able to get your low poly model done in about 30 minutes. If you have problems with points on your model, you may need to subdivide some planes to help smooth them out. •

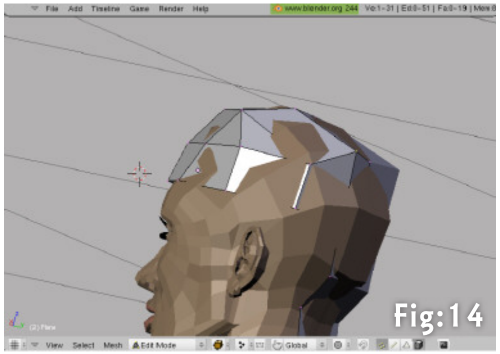


Fig:14

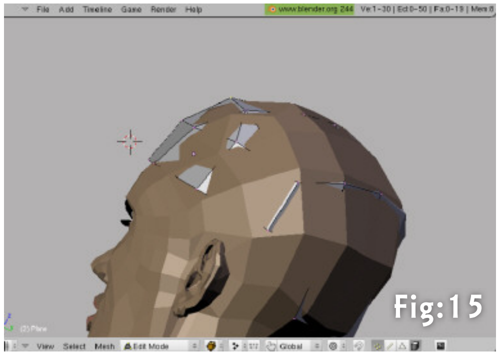


Fig:15

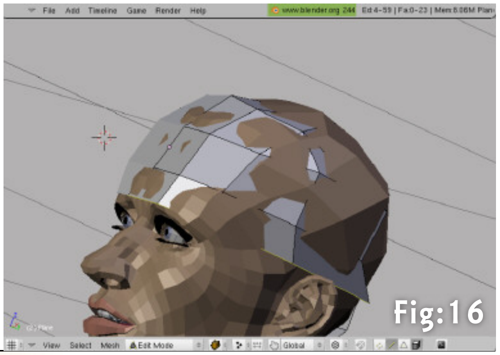
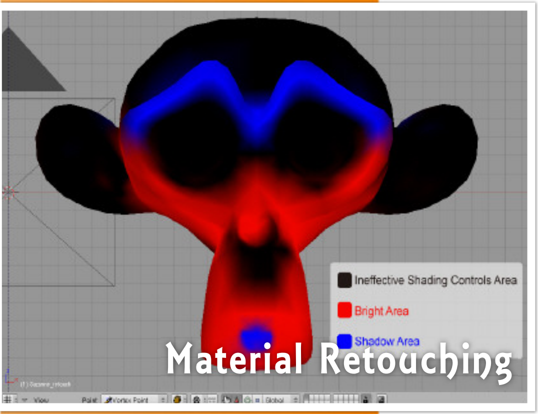


Fig:16



by Nanmo

Introduction

Material Retouching is a way to control a shaded area by using Material Nodes with Vertex Color. It is useful if you didn't get desired results in a tiny shaded area of a model. Now you could remake the model, but remaking takes time. Another way would be to use post processing as typified by use of the Composite Node to give a 2D effect, although this creates a partially incorrect 3D look. Or you can adjust shading of the model with Material Retouching in less time.

So let's see how Material Retouching works.

First, while in Vertex Paint mode, we will paint two shading control areas, one in red and one in blue. Paint your bright areas in red, paint your shadow areas in blue, and paint unimportant areas in black.

Figure.1: Vertex Color painting example.

Second, create the following nodes from the original Material as shown in the following image.

Figure.2: "Material Retouch" Material Node.

Note: To extract Red or Blue from the Vertex Colors you use an RGBCurve Node. To extract Red, you use a value of zero in the green and blue channels of the RGBCurve node. Now you can render your scene and continue to adjust your Vertex Color painting until you achieve your desired look.

Figure.3: Left:normal shading of Suzanne. Right:"Concave Retouch" shading of Suzanne. Lighting is provided by a sun lamp.

This method is based on colors because Blender 2.45 doesn't support shader influence controls.

Notice that Material Retouching only affects colors, not Ray Traced or Buffered Shadows.

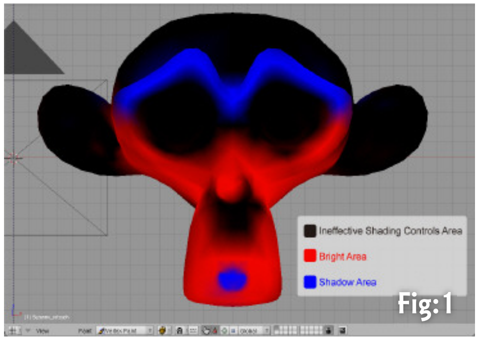


Fig:1

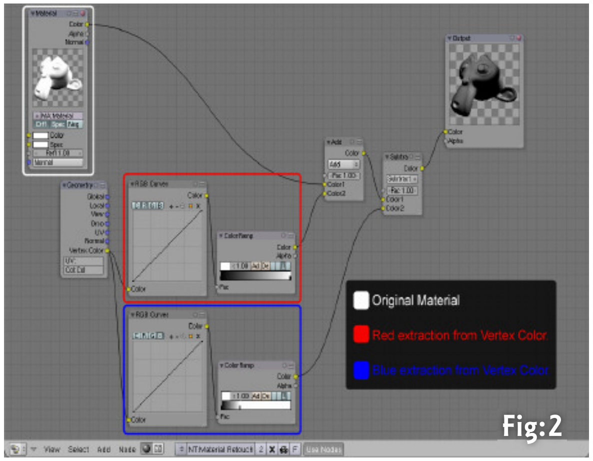


Fig:2

References:

Locally Controllable Stylized Shading (English)
http://www.olm.co.jp/en/rd/2007/07/siggraph_2007_paper.html
Locally Controllable Stylized Shading (Japanese)
http://www.olm.co.jp/b/rd/2007/07/_-locally_controllable_stylized_shading.html
Test of Toon shading retouch on 3D (Japanese,XSI)
http://artifacts.sakura.ne.jp/sakanaya/2007/10/3d.htm

Credit:

Nanmo(Japan)
http://bleble.tumblr.com/

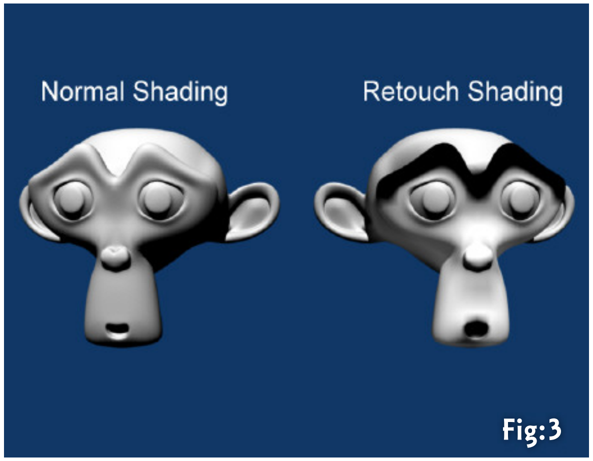
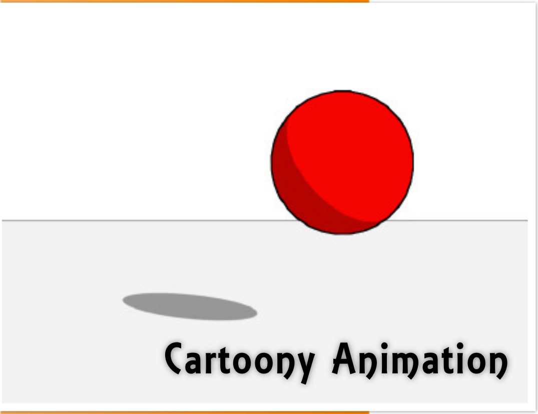


Fig:3



by Tobias Dahl Nielsen

Introduction

The thing that makes a cartoon fun to read is that they are reality*20. Cartoons are full of oversized characters that move in oversized ways, and that's what makes them interesting. Another thing that's important when creating cartoons, is making them easy to read/look at, with clear colors and sharp shadows. This tutorial will teach you how to create a little cartoony animation, using those guidelines. Happy cartooning!

Part One (Animation setup and Animation)

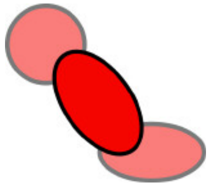
NOTE: if you just want to apply the cartoon effect to a scene you've already created, you can skip to Part Two.

The reason I've put 'animation setup' instead of 'modeling' or 'rigging' is that, to keep it simple, we'll be animating a simple sphere, though the principles of animating this sphere can be used whatever you are animating. Start by deleting the default cube and go to top view. Create a UV-Sphere and go to the Edit tab and select 'Set Smooth'. Then go to Front View and put the sphere just above the red line(the x-axis).

We will use the red line as a guideline, to show where the floor is, when we are in front view. Create a cube, scale it down a lot, and put it on top of the sphere. The cube will be our 'rig'. Select the sphere, and go to the 'Object' tab and click 'Add Constraint' and then 'Stretch To'. Write 'Cube' in the 'Target' field. Try moving the cube.

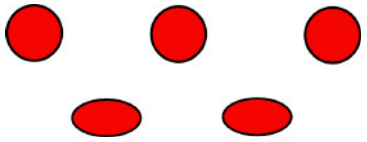
The sphere should now stretch towards the cube, while maintaining its volume (if you move the cube far away, the sphere will get thin, etc.). Go to top view and create a plane. Move it around so that it's just below the sphere. Go to camera view and scale it up until just before it exceeds the clipping. Split the 3D view near the top, and change the new area to a timeline.

Select the camera and press [ALT-R]. Rotate the camera 90 degrees by the x axis, and move it , so that it's just in front of the sphere. Go to frame 0, and click the red circle button in the Timeline area (this will 'record' the places you put the ball). Go to camera view, select both the sphere and the cube, and move them, by the x axis, to a place just outside the camera's border.



Move them both up a little. Now we're ready to go! Move the ball up and down for every tenth frame, like shown in the picture (the first ball being frame 0, the next frame 10, and so on), and use the cube to 'smash' the ball when it hits the ground, by moving it towards the ball.

Remember to move both the ball and the cube! In the frames in between (5,15,25 and so on) stretch the ball, so that one end is pointing at the previous key-pose, and the other is pointing at the next, like shown in the picture.



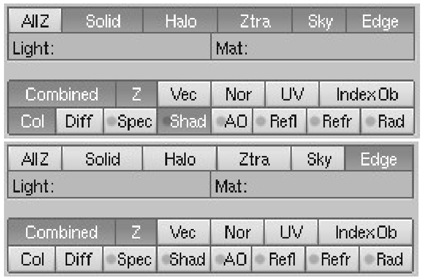
That's actually the basic principles of squash and stretch! Objects that move, stretch; objects that collide with something, squash.

NOTE: Don't expect perfect results the first time you try this! It may take some practice to get good at it!

By Tobias Dahl Nielsen

Part Two (Materials, Lighting and Node Setup)

Firstly, go to the Scene tab, and click the 'Edge' button. Then click 'Render Layers' and select 'Col' and 'Sha'. Those are the 'Color' and 'Shadow' passes (we'll need them for the node setup). Secondly, click the button below the word 'Scene:', and select 'ADD NEW'. This will create a new RenderLayer. Then disable the buttons: 'Solid', 'Halo', 'Ztra' and 'Sky'. We will need that for using 'Edge' in the node setup.

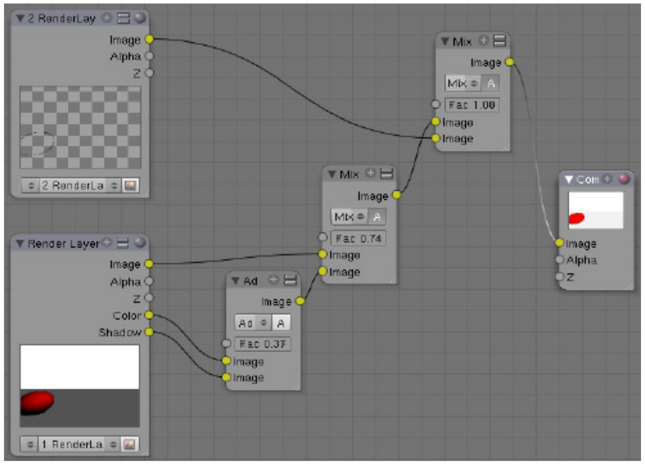

AllZ Solid Halo Ztra Sky Edge
Light: Mat:
Combined Z Vec Nor UV IndexOb
Col Diff Spec Shad AO Refl Refr Rad
AllZ Solid Halo Ztra Sky Edge
Light: Mat:
Combined Z Vec Nor UV IndexOb
Col Diff Spec Shad AO Refl Refr Rad

Materials: The materials should simply be clear colored, and without specularity (if you can do without it). I have chosen a clear red color for the sphere. Also, avoid 'Ray-Mirror', as it just doesn't mix with the cartoon style. The cube should have a completely transparent material, which shouldn't be traceable or specular.

Lighting: The lighting should consist of a single sun lamp, 'shining' from the top right of the image. NOTE: This is just for this specific project. You can use any lighting setup you want to.

Nodes: As I mentioned before, we're going to have a separate RenderLayer for the toon edge. This is actually a workaround for a Node problem I ran into, which wouldn't show the edge in some parts of the image (the ones that were in front of the ground, to be exact). This is the complete node setup:

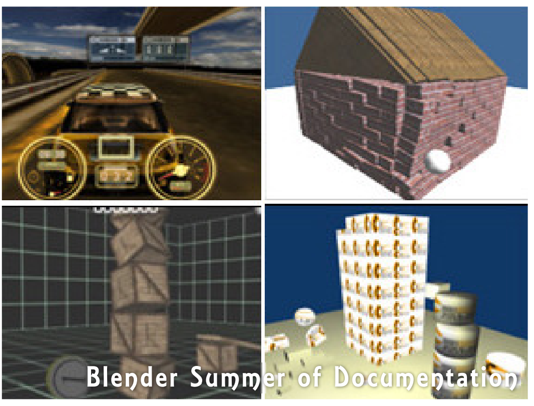
It consists of two RenderLayers (the default one, and the one we created). We 'add' the color and shadow passes from the first RenderLayer, creating the cartoony shading ('Fac' should be around 0.35). Then we 'mix' the output from that, and the image (with alpha) to get the sky displayed too ('Fac' around 0.74). At last we mix the output from that, with the image from the second layer (with alpha) creating the toon edge



('Fac' around 1). You can play around with the fac settings to get just the effect you want, however, I found that this worked best for me. Make sure that you connected the different Node's the same way I did. Now all you have to do is to select 'AVI Raw' (or whatever movie format you like) under 'Format', and click 'Anim'. That's it! Your cartoon animation is complete!

You can watch an extended version of the animation made in this tutorial here.

If you have any trouble or comments, please email me at: tobiasdn@gmail.com



by Mal Duffin

Introduction

This tutorial will show the reader how to create a simple 3D game from scratch in Blender. After completing the tutorial, the skills learned will allow readers to extend the game levels. It assumes a basic knowledge of Blender, but not of the built-in Game Engine (GE). Only aspects of Blender that are relevant to making simple games will be touched on.

The tutorial should take around 2 hours to create a working 3D game from scratch.

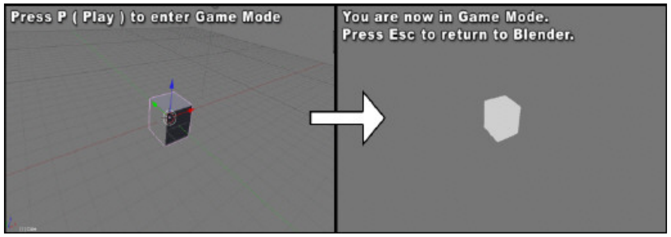
Level: Beginner to Intermediate

Learning the basics of using the Blender Game Engine

You can enter and exit Game Engine mode using the [P] and [Esc] keys.

We will now focus on the GE related features of Blender. Lets start with the most important shortcut key in Blender, the one that starts the GE. Move your mouse cursor over the 3D scene, and press [P] to Play the Game. Congratulations, you have just played your first game within Blender!!! How easy was that?

As we haven't told the GE to do anything yet, nothing will occur within the scene. Press [Esc] to return to Blender.


Press P (Play) to enter Game Mode
You are now in Game Mode.
Press Esc to return to Blender.

Setting up the default scene

Before we start, reset the scene in Blender to get back to the defaults. This can be done with any of the three methods below...

Selecting the File ›› New menu, and clicking on the Erase All option.

Press [Ctrl+X], and click on the Erase All option

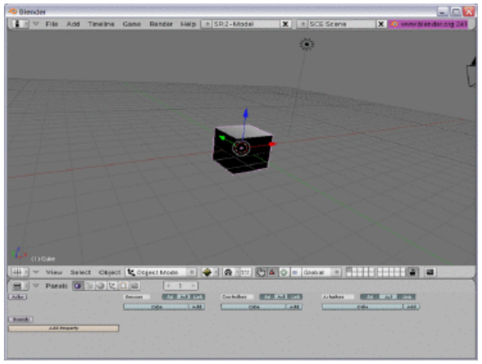
Press [Ctrl+Q] to quit Blender (versions 2.42 and prior just use [Q]. Then restart Blender manually.

Now it's time to see how well you can use Blender, using the information you learned from the previous section. Set up the 3D view in Blender similar to the screenshot. To do this, you will need to carry out the following tasks...

Rotate the scene using the middle mouse button (MMB).

Change the view into perspective mode using View ›› Perspective (or pressing [5] on the number keypad).

Add a cube and a lamp (assuming that you deleted them previously and they are not part of the default scene when Blender loads a new file).



By Mal Duffin

A very useful keypress sequence when working with the GE

One useful keypress to remember when working with the Blender GE is the one that maximizes the current 3D window.

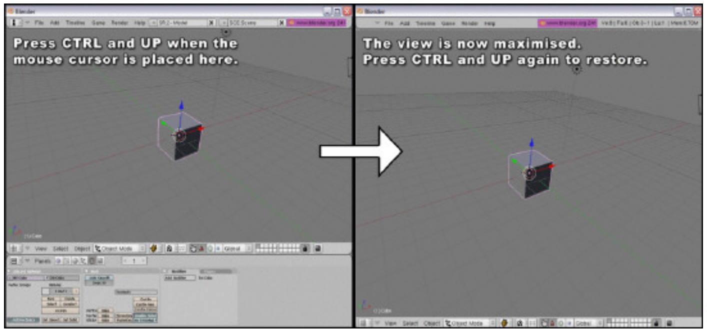
Move your mouse over the 3D window, and press [Ctrl+Up] (Cursor Up) (or Ctrl Cursor Down). This will make the current window scale to the full size of the Blender area. If you press [Ctrl+Up] again, the window will be restored to its previous size and location.

Go ahead and practice this now:

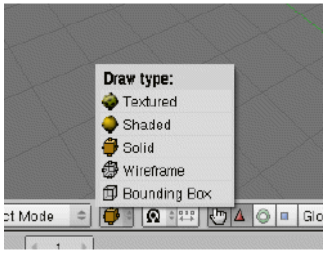
- Maximize the 3D window using [Ctrl+Up]
- Press [P] to play the current scene within the GE (nothing will happen)
- Press [ESC] to return to modelling mode
- Restore the 3D window to it's original size again using [Ctrl+ Up].

The image sequence below shows the effects of using the Maximize/Restore keypresses within the default Blender screen setup.

Maximising the Blender panels using [Ctrl+Up]


Press CTRL and UP when the mouse cursor is placed here.
The view is now maximised. Press CTRL and UP again to restore.

Choosing the correct Shading Mode (or Draw Type) for the GE



Enter GE mode again by moving over the 3D panel and press [P]. You will notice that the scene within the GE appears flat. Press [Esc] to return to Blender. Blender has a number of draw types for the viewport that are useful for different tasks. The panel to change the current shading mode is shown below...

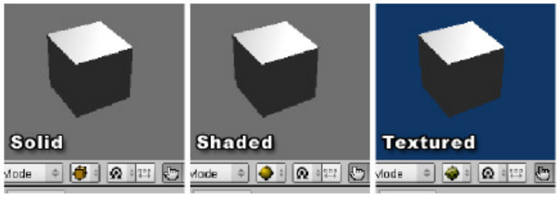
The image below shows how the basic scene looks in the GE, in the various Shading Modes

Solid mode doesn't take into account the lights in the scene

Shading mode takes into account the lights in the scene

Textured mode takes into account the lights in the scene, and also shows any textures live in the viewport. This will be as close to the actual in-game view as you can get, and should always be selected when you start a new GE project.

The best draw type for the GE is Textured. Select this Textured draw type from the list, and press [P] again. You will notice that the lighting affects the environment within the GE now, making it look more realistic. Always remember to set this option to Textured if you play your scene in the GE and it appears flat.

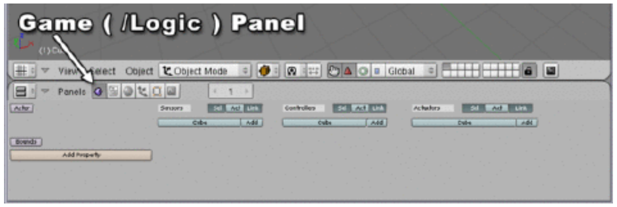


By Mal Duffin

The Main Game Logic Panel

Below the 3D window, you will see the panel that contains many different buttons for controlling different aspects of Blender.

You can view the panel related to the GE by clicking on the purple Pacman-like icon , just like below or by pressing [F4].



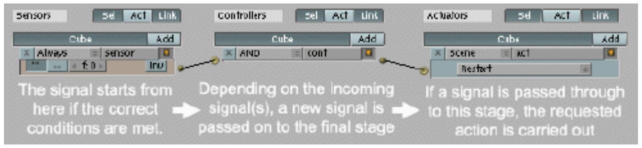
This panel is where you will control what will happen within your game.

Blender uses a visual click-and-drag system to create basic game interactions. This allows the GE to be used by 3D artists who may not have access to a coder. Blender also has a programming language, Python, which can be used to create more complex game interactions.

For the purposes of this tutorial, we will focus on the visual system for creating games. When you have grasped the basics of using the Blender GE, you can then follow more advanced tutorials showing how to implement Python scripting to create more complex games.

Visually controlling the GE - Sensors, Actuators, Controller Logic Blocks

The GE system uses Logic Blocks as a visual way to set up interactions within the game. These logic blocks can be connected together visually to allow for complex game actions to take place.


Sensors
Sel
Act
Link
Cube
Add
Always
sensor
Controllers
AND
cont
Actuators
Scene
act
Restart
The signal starts from here if the correct conditions are met.
Depending on the incoming signal(s), a new signal is passed on to the final stage
If a signal is passed through to this stage, the requested action is carried out

There are three different types of Logic Blocks - Sensors, Controllers and Actuators - each with a number of different subtypes.

Sensors

A Sensor will detect some form of input. This input could be anything from a keypress, a joystick button or a timer that triggers every single screen update (or frame) of the game.

Controllers

Controllers are used to link Sensors to Actuators. They allow for some more complex control over how sensor and actuators interact with each other.

Actuators

An Actuator will actually carry out an action within the game. This can include moving an object within a scene, playing an animation, or playing a sound effect.

By Mal Duffin

Setting up a basic Sensor, Controller, Actuator Logic Block sequence.

We will now set up a very basic system within the game panel by adding and connecting a sensor, controller and actuator together.

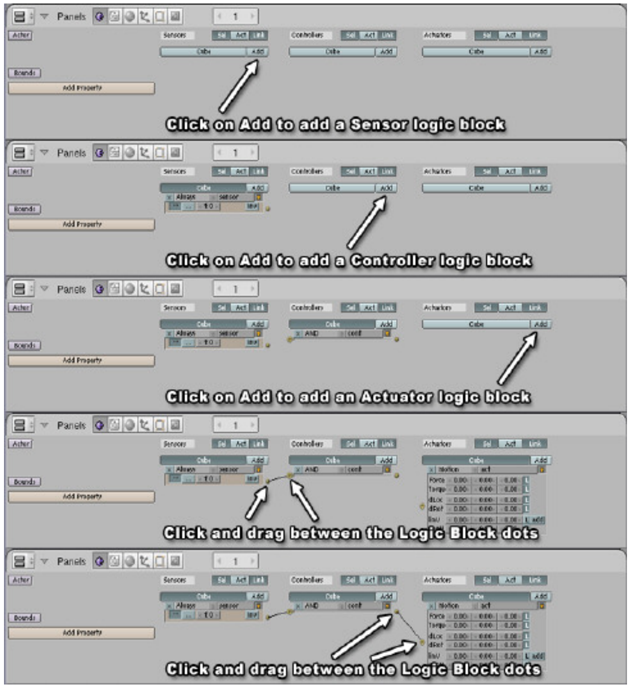
GE Panel: Make sure the Game Logic panel is visible (click on the purple pacman in the Buttons window or press [F4]), and re-select the cube within the 3D scene

Below each of the 3 main sections, you will see the selected object's name, and an Add button. Click this Add button once for each of the 3 sections: Sensor, Controller and Actuator.

We will now connect this system together. Click and drag from the socket (small circle) at the end of the first sensor to the socket at the start of the controller. Then click and drag from the socket at the end of the controller to the socket at the start of the actuator.

The image sequence on right shows the steps involved in setting up a simple Sensor, Controller, Actuator chain and connecting them together.

Press [P] to now play the game. You will notice that, although we have added some control to the GE, nothing seems to happen again. This will be explained in the next section. Press [Esc] to return to Blender.


Click on Add to add a Sensor logic block
Click on Add to add a Controller logic block
Click on Add to add an Actuator logic block
Click and drag between the Logic Block dots
Click and drag between the Logic Block dots

Breaking down the events in the GE system

We will now look into what is happening with our newly created GE system. The sensor is an Always timer. This will send out a signal every single frame keeping the linked controller continuously activated thoughout the duration of the game.

The controller is an AND. When it has just one active sensor input (like in the current case), it will automatically call the connected Actuator.

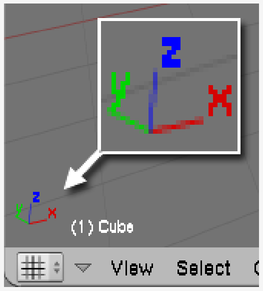
The Actuator controls the Motion aspects of the selected object. Because of the Always sensor connected to the controller, this actuator will be called every single frame.

If we press [P] now, the Motion actuator is being called every single frame, but because all of its values are set to zero, it will not move the object within the GE. Press [Esc] again to return to Blender.

By Mal Duffin

Keeping Track of Directions

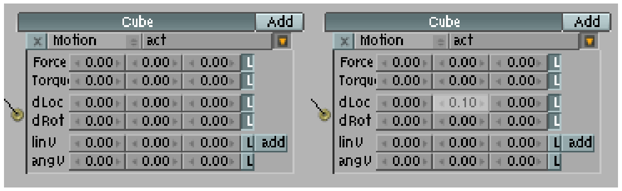
To keep track of what direction X, Y and Z are in, keep an eye on the visual axis in the bottom left of the 3D view.



Making the default cube move, using no physics

We will initially use direct manipulation to move the cube within the scene. Later on, we will set up a similar scene, but will use physics to move the cube around the environment. By using the built-in physics engine (called Bullet), more complex scene interactions such as collisions and gravity will be handled automatically.

Have a look at the Motion actuator, especially the 3 numerical boxes beside the dLoc label. Each of the 3 boxes in this dLoc area can be used to specify a change to the location of the object along the X, Y or Z axis respectively.



Change the middle dLoc numeric value (Y axis, or forward) to be 0.10.

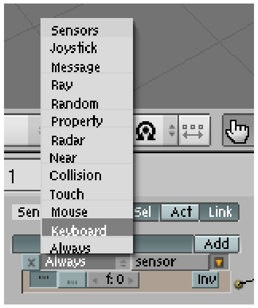
Now press [P]. You will notice that the cube continually moves along the Y axis. Press [Esc] to return to Blender. Press [P] again and you will notice that the exact same sequence of events in the GE occurs again. Press [Esc] to return to Blender again.

To recap on what is happening in the GE - the timer (Sensor) sends out a signal every frame to the Cube's Controller. This then sends the signal to the Cube's Actuator, which will in turn call the Motion object, which moves the cube object 0.1 units in the Y axis. Since it is called every single frame, it looks like the cube is continually moving. If you let this game run, the cube will eventually glide off to infinity. This is about as fun as watching grass grow, so read on!

Controlling the cube via the arrow keys

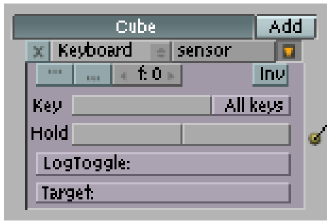
In the example on right, the timer is responsible for moving the cube forward. We will change this now so that a keyboard press controls it.

On the Always sensor, click on the ◇ beside the Always label. This will bring up a list of the available sensor types (see fig below).

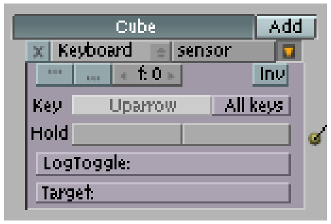


Select Keyboard from the list. The Sensor will now change to show the Keyboard options.

This new sensor panel will allow us to choose a key that needs to be pressed before a signal is sent to the controller, which in turn will pass it on to the actuator. Click on the button area beside the Key label, and when it changes to say press a key, press the [Up arrow] key.



Press [P] to play the game again. The cube will no longer move automatically. Press the [Up] key, and it will start to move forward. When you stop pressing the [Up] key, the cube will stop moving. Press [Up] again to move the cube again. Press [Esc] to return to Blender.



By Mal Duffin

Adding in additional keyboard controls

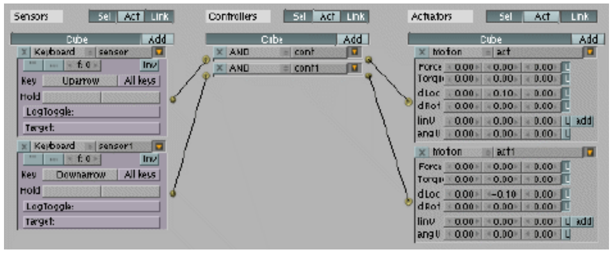
We will now add in the ability to move the cube backwards, as well as rotating it, to allow you to move it around the 3D environment.

To add the ability to move backwards, we will add in a new (but nearly identical) set of GE logic bricks. Click the Add button again on the Sensor, Controller and Actuator areas, to create 3 new logic bricks on the GE panel. Connect these as before.

NOTE - You might want to use the [Ctrl+Up] key to maximise and restore the game logic panel when working in it, as it will become cluttered very quickly.

With the newly added sensor, change it to keyboard type, and set it to use the [Down] arrow key. In the newly added Motion actuator, change the Y value (2nd column) of the dLoc area to be -0.1.

Press [P] to play the game. If the logic blocks are set up correctly, you should now be able to move the cube forward and



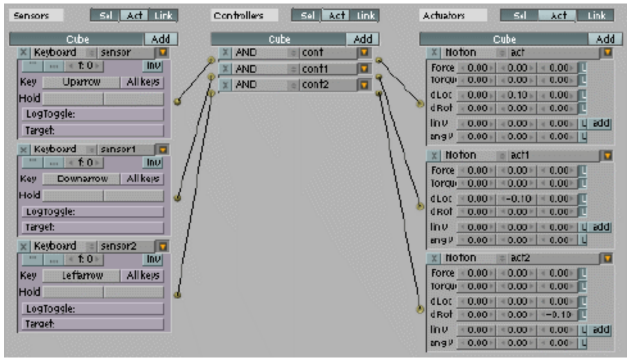
```
Sensors    Sel Act Link      Controllers  Sel Act Link    Actuators   Sel Act Link
Cube                 Add     Cube                 Add     Cube                        Add
Keyboard   sensor            AND   cont                   Motion   act
  f: 0          Inv          AND   cont1                  Force   0.00   0.00   0.00  L
Key  Uparrow   All keys                                   Torque  0.00   0.00   0.00  L
Hold                                                      dLoc    0.00   0.10   0.00  L
LogToggle:                                                dRot    0.00   0.00   0.00  L
Target:                                                   linV    0.00   0.00   0.00  L add
Keyboard   sensor1                                        angV    0.00   0.00   0.00  L
  f: 0          Inv                                       Motion   act1
Key  Downarrow All keys                                   Force   0.00   0.00   0.00  L
Hold                                                      Torque  0.00   0.00   0.00  L
LogToggle:                                                dLoc    0.00  -0.10   0.00  L
Target:                                                   dRot    0.00   0.00   0.00  L
                                                          linV    0.00   0.00   0.00  L add
                                                          angV    0.00   0.00   0.00  L
```

backward just by using the [Up] and [Down] arrow keys.

To complete this "no physics" part of the tutorial, we will add in the ability to rotate the object as well, so that you will be able to "drive" your model around the 3D environment.

Add in 3 new logic blocks as before, connect them, and change the sensor type to keyboard. Set the keyboard entry to use the [Left] arrow key.

Now, in the Motion Actuator, we will set it to rotate the cube. To do this, we will rotate the object around its Z axis. In the dRot area, change the Z value (3rd column) to be -0.1.



```
Sensors    Sel Act Link      Controllers  Sel Act Link    Actuators   Sel Act Link
Cube                 Add     Cube                 Add     Cube                        Add
Keyboard   sensor            AND   cont                   Motion   act
  f: 0          Inv          AND   cont1                  Force   0.00   0.00   0.00  L
Key  Uparrow   All keys      AND   cont2                  Torque  0.00   0.00   0.00  L
Hold                                                      dLoc    0.00   0.10   0.00  L
LogToggle:                                                dRot    0.00   0.00   0.00  L
Target:                                                   linV    0.00   0.00   0.00  L add
Keyboard   sensor1                                        angV    0.00   0.00   0.00  L
  f: 0          Inv                                       Motion   act1
Key  Downarrow All keys                                   Force   0.00   0.00   0.00  L
Hold                                                      Torque  0.00   0.00   0.00  L
LogToggle:                                                dLoc    0.00  -0.10   0.00  L
Target:                                                   dRot    0.00   0.00   0.00  L
Keyboard   sensor2                                        linV    0.00   0.00   0.00  L add
  f: 0          Inv                                       angV    0.00   0.00   0.00  L
Key  Leftarrow All keys                                   Motion   act2
Hold                                                      Force   0.00   0.00   0.00  L
LogToggle:                                                Torque  0.00   0.00   0.00  L
Target:                                                   dLoc    0.00   0.00   0.00  L
                                                          dRot    0.00   0.00  -0.10  L
                                                          linV    0.00   0.00   0.00  L add
                                                          angV    0.00   0.00   0.00  L
```

Press [P] to play the game. When you click the [Left] key, the cube will rotate. When you press the [Up] arrow key, the cube will move forward in that direction. Press [Esc] to return to Blender, and we will add in the ability to rotate the other way.

As before, add in 3 new logic blocks, connect them, and change the sensor type to keyboard.

Set the keyboard entry to use the [Right] arrow key, and the Z value (3rd column) in the dRot section in the Motion actuator to be 0.1.

Press [P] again to run the game. You can now "drive" the cube around the 3D environment using the arrow keys. Note that you can add this game logic to any model in Blender (no matter what shape or size it is), and it will move around just like the cube. Press [Esc] to return to Blender.

By Mal Duffin

Some GE house keeping

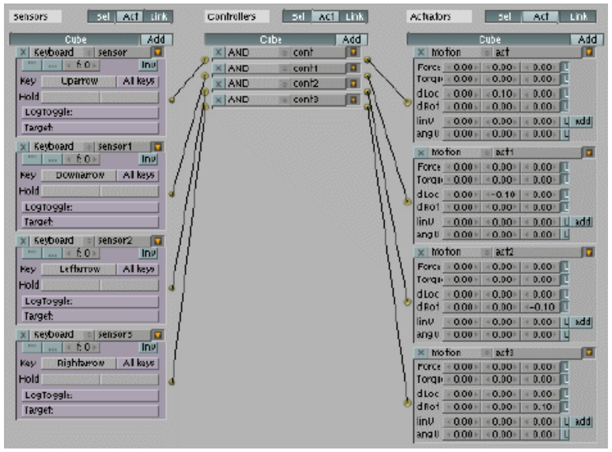
We will now name some of the sensors and minimise them, to keep the game panel as uncluttered and legible as possible. This won't affect what the actual logic does, but it will help keep the game logic area more manageable.

Press [Ctrl+Up] in the game panel to make it full-screen.

In the first sensor panel, next to where it says Keyboard, you will see an area where the text Sensor appears. This can be changed to something more descriptive. Change this text to "up key". Next, press the arrow button beside this to minimise the Sensor panel.

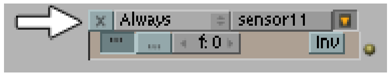
Repeat this for the other keys. Also, do the same in the Actuators, giving them a descriptive title (for example "move forward", "turn left", etc...).

Press [Ctrl+Up] again to restore the game panel to its original size.



As you can see, having the various blocks named make it a lot easier to figure out what is going on.

Removing logic blocks and connections



To remove logic blocks, press the X button at the top left corner of the logic blocks. To remove the connections between logic blocks, move the mouse over the line connecting the blocks and press the [Del] key.

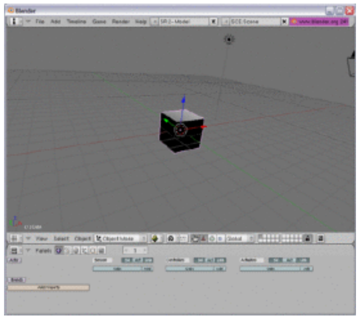
Making the default cube move, using physics

One of the most powerful features of the Blender GE is the built-in physics engine (Bullet).

By using forces to move an object around the scene, the physics system will automatically handle complex interactions such as resolving collisions with other objects in the scene. For many game types, using Physics will solve more complex issues, but it does require slightly more work to set it up.

We will now create a basic game from scratch using physics. The scene will consist of a sphere, which we will move around the scene using physical forces.

Before we start, reset the scene in Blender to get back to the defaults, and set up the 3D view in Blender to look like the image on right.



3D WORKSHOP: Learning the Blender Game Engine

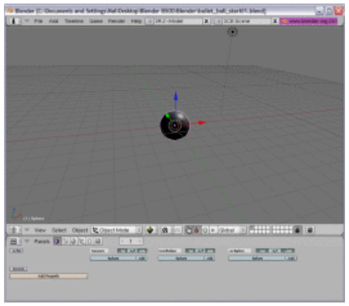
By Mal Duffin

Set up Blender to look like this for the following tutorial

- Select the File ›› New menu, and click on the Erase All option.
- Rotate the scene using the middle mouse button MMB
- Change the view into perspective mode using View ›› Perspective (or Keypad [5])
- Change the 3D view to use the Textured shading mode (/Draw Type) or press [Alt+Z]
- Open the Game (/Logic) panel

Setting up the game scene To start creating the game scene, delete the cube by pressing [Del].

- Add in a sphere using Add ›› Mesh ›› UVSphere (pressing the [spacebar] also brings up this menu).

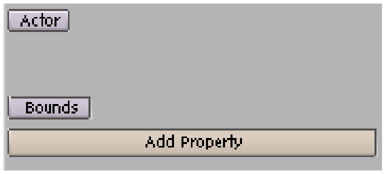


- When asked, select the default 32 segments and 32 rings as the setup of the sphere.
- Exit Edit mode straight away on the sphere by pressing [Tab], to return to Object mode.
- Also, press [Alt+R] to clear the rotation on the object (The object must still be selected).

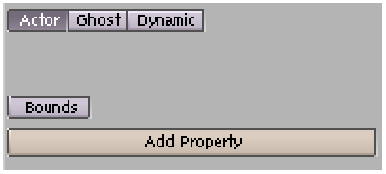
Making a model physical within the GE

In the game panel, you will see an unselected Actor button.

Click on this button, and a number of other features will become available for the selected model.

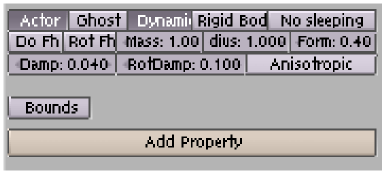


Select the Dynamic option - This will tell the GE that the model is a physical object. More options will then also appear.



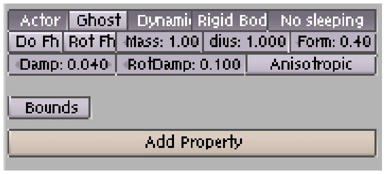
Select the following options...

- Rigid Body The physical object will automatically rotate correctly using the GE physics engine. If this isn't selected, the object will be able to move, but not rotate.



Mass: 1.00 | dius: 1.000 | Form: 0.40 | Damp: 0.040 | RotDamp: 0.100

- No sleeping - The physical object will never be de-activated (also known as sleeping).



Mass: 1.00 | dius: 1.000 | Form: 0.40 | Damp: 0.040 | RotDamp: 0.100

Now, press [P] to enter the GE. You will notice that, even though we haven't added in any game engine logic blocks, the ball starts moving. This is because gravity is affecting the ball, so it falls down. This illustrates one of the features of interacting within a physical world. Press [Esc] to go back to Blender. Press [P] again and you'll notice that the same thing happens. Press [Esc] to return to Blender.

By Mal Duffin

We need to add something for the ball to fall onto, for example a ground object.

- Add a plane model to the scene using Add ›› Mesh ›› Plane.
- Press [Tab] to exit Edit Object mode, and return to Object mode.
- Press [Alt+R] to reset the rotation on the model.

Use the 3D Transform Manipulator to move the plane underneath the sphere, and press [P]. You will notice that the physical sphere now falls due to gravity, but will land on the plane below it and come to a rest.

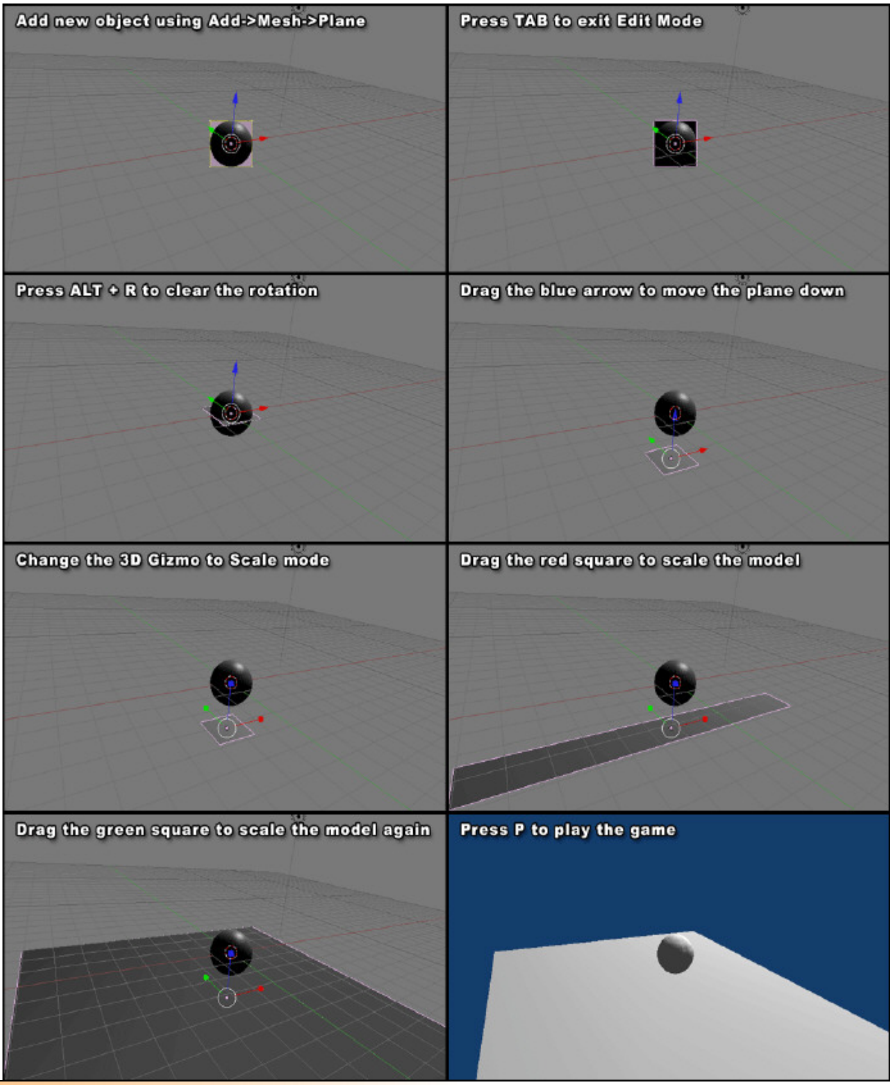
We will now scale the plane up [S], so that we have plenty of room to move the ball around within.

Change the 3D transform manipulator from move mode into scale mode.

Grab the scaling handles and size the plane so that it is around 10 times larger in the X and Y directions.

The image sequence on right illustrates the sequence of steps indicated above.

Adding and placing a ground object
(See images on right)


Add new object using Add->Mesh->Plane
Press TAB to exit Edit Mode
Press ALT + R to clear the rotation
Drag the blue arrow to move the plane down
Change the 3D Gizmo to Scale mode
Drag the red square to scale the model
Drag the green square to scale the model again
Press P to play the game

By Mal Duffin

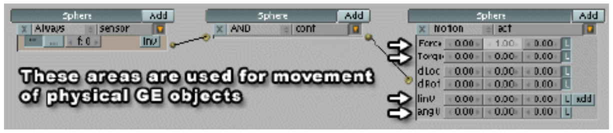
Moving the physical object within the GE

We will now apply physical forces to the sphere, to make it move around the 3D environment.

IMPORTANT - Make sure the Sphere model is selected. If you have just created the ground plane model, it will still be selected, so you will have to right-click on the sphere to re-select it.

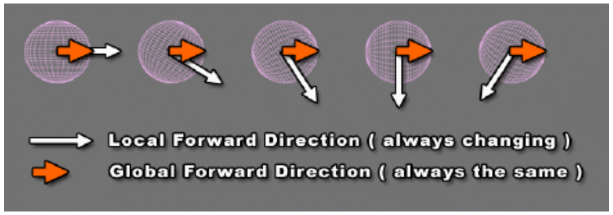
Add in a new sensor, controller and actuator object in the game panel, and connect them together by clicking and dragging between the dots.

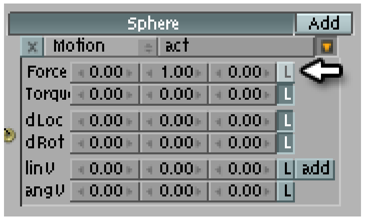
In the motion actuator, we will need to set the values in the Force section in order to move the physical object around the scene. In the force section, change the Y value (2nd column) to 1.


These areas are used for movement of physical GE objects

Now press [P]. You will now see the ball falling onto the ground / plane object, and begin to roll.

After the ball rolls a certain distance, you will notice that it starts to roll back on itself. This is due to the fact that we are applying the force locally, along the balls Y axis. As the ball rotates, it's Y axis also rotates, as seen below. Press [Esc] to return to Blender.


Local Forward Direction (always changing)
Global Forward Direction (always the same)


```
Sphere                         Add
Motion        act
Force   0.00   1.00   0.00   L
Torque  0.00   0.00   0.00   L
dLoc    0.00   0.00   0.00   L
dRot    0.00   0.00   0.00   L
linV    0.00   0.00   0.00   L  add
angV    0.00   0.00   0.00   L
```

In order to fix this, we will change the force movement from local to global. To do this, click on the L at the right of the Force section to deselect it. Press [P] again, and you will see that the ball continually moves in the correct direction.

You may not have noticed how the physics engine is working in the background.

As we apply a sideways force to the ball, it will start to roll. This rolling is caused by friction between the surface of the ball and the ground.

Also, as the ball reaches the end of the plane, it will realistically tip off the end of the object, and continues falling.

These are a few of the advantages of using physics within the GE.

Press [Esc] to return to Blender.

Controlling the sphere using the arrow keys

We will now take control of the sphere using the arrow keys.

Change the Always sensor to Keyboard, and set it to use the [Up] arrow key.

Now, add and connect additional sensors, controllers and actuators, where...

By Mal Duffin

We will now apply physical forces to the sphere, to make it move around the 3D environment.

IMPORTANT - Make sure the Sphere model is selected. If you have just created the ground plane model, it will still be selected, so you will have to right-click on the sphere to re-select it.

Add in a new sensor, controller and actuator object in the game panel, and connect them together by clicking and dragging between the dots.

In the motion actuator, we will need to set the values in the Force section in order to move the physical object around the scene. In the force section, change the Y value (2nd column) to 1.

Now press [P]. You will now see the ball falling onto the ground / plane object, and begin to roll.

After the ball rolls a certain distance, you will notice that it starts to roll back on itself. This is due to the fact that we are applying the force locally, along the balls Y axis. As the ball rotates, it's Y axis also rotates, as seen below. Press [Esc] to return to Blender.

In order to fix this, we will change the force movement from local to global. To do this, click on the L at the right of the Force section to deselect it. Press [P] again, and you will see that the ball continually moves in the correct direction.

You may not have noticed how the physics engine is working in the background.

As we apply a sideways force to the ball, it will start to roll. This rolling is caused by friction between the surface of the ball and the ground.

Also, as the ball reaches the end of the plane, it will realistically tip off the end of the object, and continues falling.

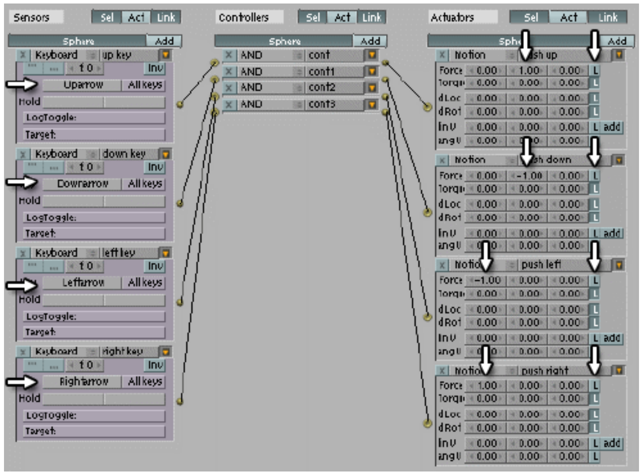
These are a few of the advantages of using physics within the GE. Press [Esc] to return to Blender.

Controlling the sphere using the arrow keys

We will now take control of the sphere using the arrow keys.

Change the Always sensor to Keyboard, and set it to use the [Up] arrow key. Now, add and connect additional sensors, controllers and actuators, where...

- The [Down] keyboard sensor controls the motion actuator with a -1 in the Y (2nd) location of the Force section (with L for Local de-selected.)
- The [Left] keyboard sensor controls the motion actuator with a -1 in the X (1st) location of the Force section (with L for Local de-selected.)
- The [Right] keyboard sensor controls the motion actuator with a 1 in the X (1st) location of the Force section (with L for Local de-selected.)



```
Sensors (Sphere)
Keyboard  up key     Uparrow     All keys
Keyboard  down key   Downarrow   All keys
Keyboard  left key   Leftarrow   All keys
Keyboard  right key  Rightarrow  All keys

Controllers (Sphere)
AND  cont
AND  cont1
AND  cont2
AND  cont3

Actuators
Motion  push up
Force  0.00   1.00  0.00  L
Motion  push down
Force  0.00  -1.00  0.00  L
Motion  push left
Force -1.00   0.00  0.00  L
Motion  push right
Force  1.00   0.00  0.00  L
```

By Mal Duffin

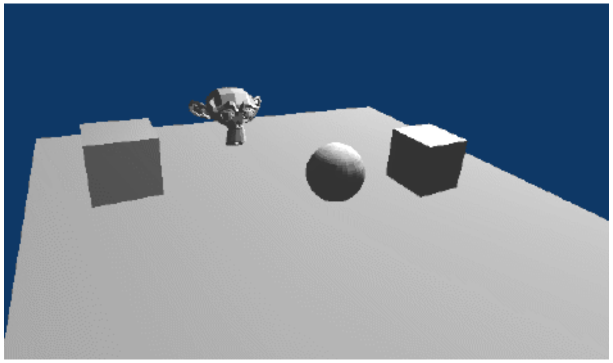
When you press [P] again, you will be able to control the ball and move it around the ground plane. Press [Esc] to return to Blender.

To make the game panel screen more legible, give the various sensors and actuators suitable labels, such as "up key" for the first sensor, and "push up" to the first actuator.

Also, try changing the values in all of the motion actuators from 1 to 2, and from -1 to -2. Press [P] now, and you will see the difference in the speed of the ball when using the arrow keys.

Adding some obstacles into the level

Add a cube to the environment, using Add ›› Mesh ›› Cube. Press [Tab] to exit Edit mode and return to Object model. Press [Alt+R] to clear the rotation on the object, and drag the arrows of the 3D transform manipulator to place the box somewhere on the surface of the plane object. Repeat this step to add a few more objects to the surface of the plane, including cylinder and monkey objects.



Now, press [P]. You will see that the ball will automatically collide and bounce off the objects that you just added. Again, this is one of the advantages of using physics within the GE. Press [Esc] to return to Blender. You might want to try adding in planes and scaling them to make ramps and jumps.

If you have experience with editing models in Blender, you can spend some time now creating a more complex level layout. If you don't have experience modelling in Blender, hopefully this tutorial will have given you an interest in Blender and learning more, including how to edit models. You can see some links to tutorials on editing at the end of this tutorial.

Making some of the objects physical

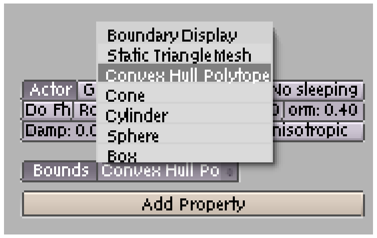
Select the cube that you have added into the new scene. In the Game Panel, select the following options, the same ones that were set for the main sphere (except that No Sleeping is not selected this time, to allow the objects to settle and rest / sleep - when an object is sleeping, it takes less time to compute within the physics system)

- Actor
- Dynamic
- Rigid Body

Now, press [P] to play the game, and move the main sphere into the cube. You will see that the cube now gets knocked out of the way.

However, the box moves in a very odd way - it actually moves as if it was a sphere.

Currently the physics system will assume that a newly added rigid body will have a spherical collision shape. Press [Esc] to return to Blender.



You will notice that there is a Bounds button below the Actor area. Click this button, and an additional dropdown will appear, with the default setting of Box.

In the case of the cube, a Box collision type will work fine. However if you have a more complex shape, you will want to select the Convex Hull Polytope option.

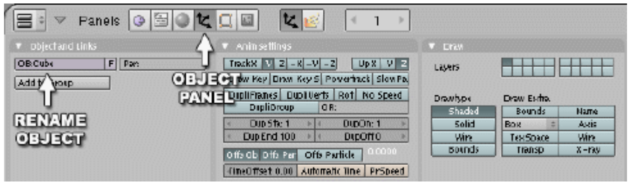
Select some of the other objects you have added to the scene, and carry out the same steps as above, selecting Convex Hull Polytope as the bounds type of the object.

Press [P] again to play the current level, and roll into the various physical objects to move them out of the way. Press [Esc] to return to Blender.

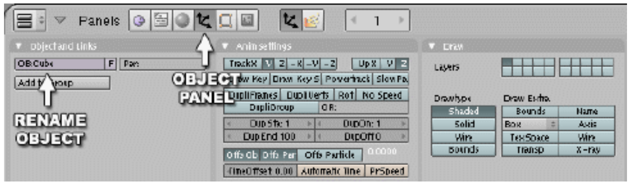
Renaming objects

When you add a new object, Blender will assign it a default name (e.g. Cube, or Cube.001 if Cube already exists).

It is good practice to rename your objects using more relevant terms, such as player, crate, pickup, etc... This will make your scene easier to understand when it gets more complex, as well as keeping it understandable for other people viewing it. To rename an object, you can select the Object panel, and change the name within the OB: area.



You can also rename the object within the Editing panel.



Completion of the basic GE tutorial

Congratulations on completing the basic Blender Game Engine tutorial! You should now have a general overview of the basics of using the GE. You will have practical experience of...

- Connecting sensors, controllers and actuators in the game panel
- Using the motion actuator to move objects directly
- Using the motion actuator to move objects using physical forces
- Taking keyboard control of game objects
- Creating a simple 3D game scene
- Making new objects physical within the GE

By Mal Duffin

By Mal Duffin

With the skills you have learned so far, you will be able to extend this simple environment as you learn more about modelling within Blender.

At this stage, you might want to recreate the final scene again, starting from scratch, to see how far you can get without reading the instructions. If you can recreate it all from memory, you are on your way to becoming a true Blender GE power user!

The next few additional areas below will cover some more complex issues, such as making the ball jump, and adding materials to the scene.

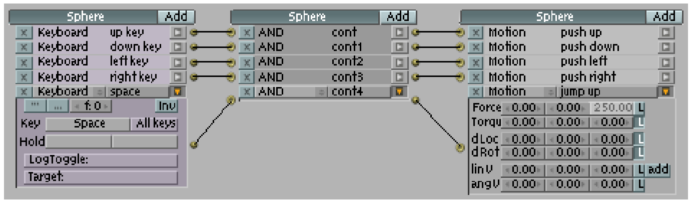
Creating more complex game levels and interactions :

Making the ball jump

We will make the ball jump when the [Space] key is pressed.

Add a new sensor, controller and actuator to the scene and connect them. Change the Always sensor to a keyboard sensor, and set it up to use the [Space] key.

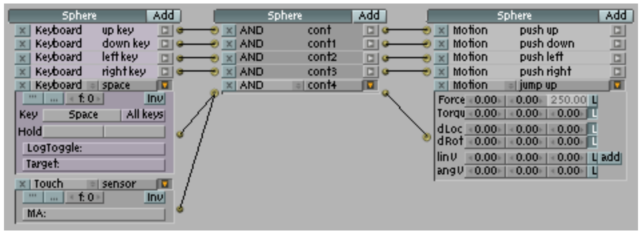
In the Force section of the Motion actuator, set the 3rd value (Z / up) to be 250. Also, click the L to turn it off, so that the force is applied along the global direction, rather than the local Z direction.



This will give the ball quite a bit of a knock, making it look like it is jumping up in the air.

Press [P], and in the game, tap the Space bar. You will notice that the ball will jump directly up in the air. However, if you continually hold down the Space button, the ball will continually go up in the air.

We need to add in an additional rule - the ball can only jump when it is touching the ground. To do this, we will use the Touch sensor. Add just an additional sensor to the scene, and change it to a Touch type. Now, connect this sensor to the controller that the Space keyboard sensor is connected to.



The controller will only send a signal to the connected motion actuator (for the jump action) when BOTH Space is pressed, AND the ball is colliding with another object, such as the ground.

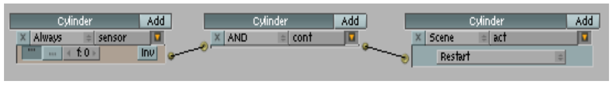
Restarting the game when a goal is reached

When the ball touches a certain object, we will set it to restart the level.

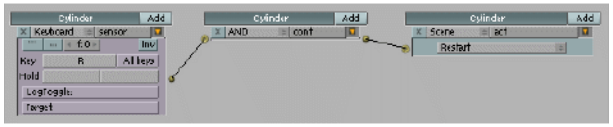
- Add a new object into the scene (in this case we use a cylinder), and place it somewhere near the main sphere (but not touching it).

By Mal Duffin

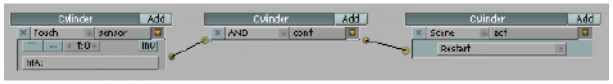
- Make sure the new object is selected (and not the main sphere).



- Open the Game Logic panel, add a sensor, controller and actuator and connect them together.
- Change the type of Actuator to Scene. The default setting of the Scene actuator is Restart.
- Now, press [P] to play the game. It will look like nothing is happening. That's because the Always sensor is continually triggering the Restart Scene sensor.
- Press [Esc] to exit the game.
- Change the sensor type to Keyboard, and set it up to use the [R] key.
- Press [P] again to play the game, and you will be able to move the sphere around.



- Pressing [R] will restart the game, and allow you to continue.
- Press [Esc] to exit the game.
- Finally, change the sensor type again to Touch.

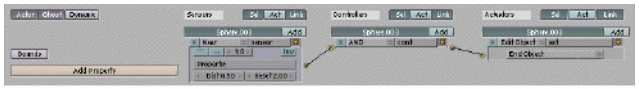


- Press [P] again to play the game. The scene will restart now when you run into the cylinder with the sphere.
- Press [Esc] to exit the game.

This illustrates some basic game engine scene management. You could have the game go to a different scene (not covered in this tutorial) that might contain a game win or game lose sequence, or the scene might be an additional level of the game.

Collecting pickups within the level

- We will now add in an object that the player can collect when they move close to it.
- Add a sphere to the scene, and place it at a reasonable distance from the main sphere.
- Add and connect a Sensor, Controller and Actuator.
- Change the Sensor type to Near
- Change the Actuator type to Edit Object.
- For Edit Object, change it from Add Object to End Object.
- Also, activate both the Actor and the Ghost buttons. The Ghost object means that other objects (including the main player) will NOT be able to collide with the object.



- Press [P] to play the game. Now when you move near the object, it will disappear (by ending itself).
- Press [Esc] to return to Blender.

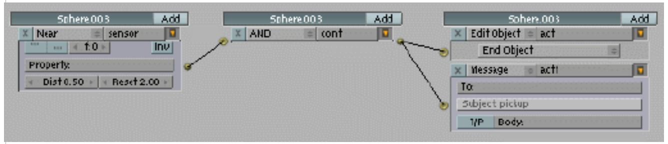
By Mal Duffin

Counting the collected objects

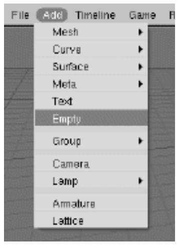
When a pickup is collected, we will update a property value to reflect the total number of collected items so far.

We will use the Message actuator to send a signal to another object in the scene, which will have a Message sensor. This sensor will trigger an actuator that will increase the value of a property.

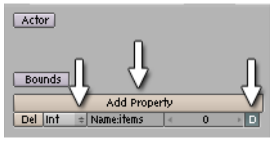
- Add another actuator to the pickup object, and change its type to Message.
- Connect this Message actuator to the existing Controller.
- Change the subject name to be something like "pickup". It's important to remember this name, as it will be required for the Message Sensor later on.



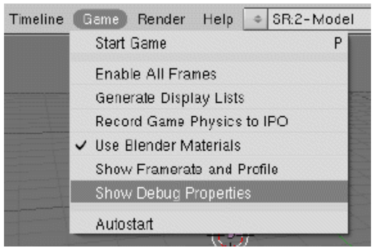
When the near sensor is activated, both of these actuators will be triggered and the message will be sent to all of the objects in the scene.



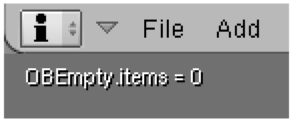
- We will use another object to store the pickup count information. For this task, an Empty object is useful - this is an object that will exist in the scene, but as it doesn't have any geometry it will not be visible in the game engine.
- Add an Empty object to the scene (Adding an Empty object.).
- When this object is selected, click on the Add Property button in the Game Logic panel (See Add Property image).



- Change the name of the newly added object to items, and change the type from Float to Int (Integer, or whole number, e.g. 0, 1, 2, 3). This is where we will store the number of items collected.



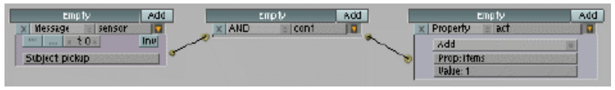
- Finally, press the D (for "Debug") button to the right of the property. This will allow you to see the value of the property in-game. In order to see the debug properties in-game, select the Show Debug Properties menu option.



OBEmpty.items = 0

- Press [P] to play the game now, and you will notice some text in the top left corner of the 3D screen, showing the value of the items property (currently set to 0).
- Press [Esc] to return to Blender.
- On the Empty object, add in a Sensor, Controller and Actuator Logic Block sequence, and connect them together.
- Change the Sensor type to Message, and set the subject name to "pickup" (the same name that you placed in the Message sensor on the "pickup" object).
- Change the Actuator type to Property, and change Assign to Add.
- Change the Prop name to "items" (the name of the property to add to), and set the value to 1.

By Mal Duffin



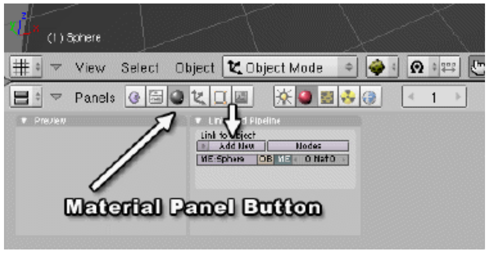
Now, press [P] to play the game, and collect a pickup - you will notice that, when you collect it, the value of the property increases. Press [Esc] to return to Blender.

You can use the result of this property to affect your game, such as restarting or going to a new scene when a certain number of pickups are collected.

Adding color to the levels using Materials

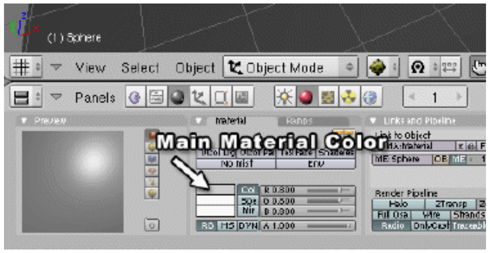
Up until now, the added objects have used the default gray color. In order to change the basic look of the scene, we will now change the colors of the models in the scene by creating new materials for the them.

- Open the Materials panel by clicking on the gray sphere on the panel ( ) or by pressing [F5], as shown below.
- Select the main sphere model in the 3D view with a right click RMB .

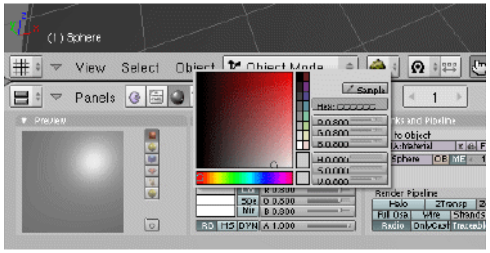


- Click on the Add New button. This will add a new material to the sphere model. A more complex set of panels will appear. For now, we will just change the color of the material.

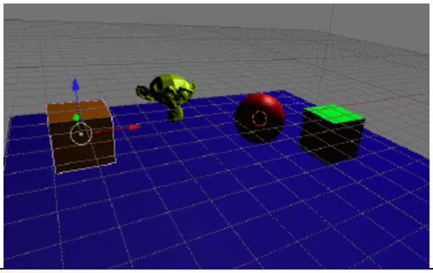
- Click on the area beside Col, which indicates the main material color.



- A color picker will then appear. Use it to choose a red color, and then move the mouse cursor away from the picker. The sphere will appear red in the 3D viewport.



Repeat this process for some of the other models in the scene until they are all different colors.



NOTE

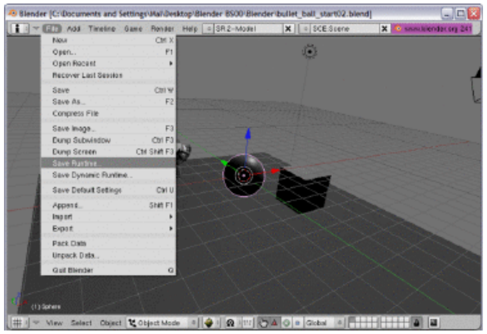
You may need to include a few other files along with that executable. Copy the file to a new folder and run it. If it gives an error that a file is missing, copy that file (probably a .dll file) into the same folder. The .dll's are usually located in Blender's default installation directory. Continue this process until the game runs. You will then be able to distribute those files to your colleagues. Just make sure the .dll's and the game's .exe reside in the same folder.

By Mal Duffin

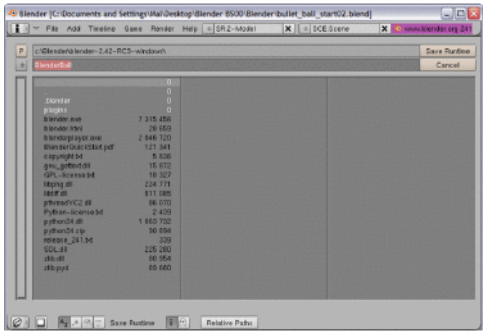
Making a stand-alone version of the game

Blender allows you to create a stand-alone version of your game to distribute to colleagues, without them having to install Blender. Your game will automatically run when the program is run.

In Blender, select File ›› Save Runtime.



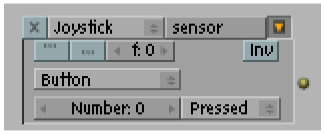
In the Save screen, enter a name for the game executable (for example ball_game). This will create a ball_game executable in that folder, which you can distribute to your friends.



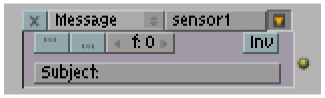
Overview of all of the Sensor, Controller and Actuator Logic Blocks

Sensors

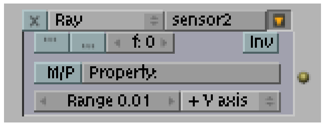
Joystick - Triggers when either a joystick button is pressed, or when a joystick is moved along a certain direction (left/right, up/down, etc).



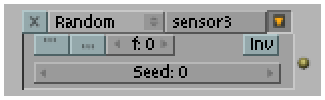
Message - Triggers when a message is received. You can send messages to other objects using a Message Actuator.



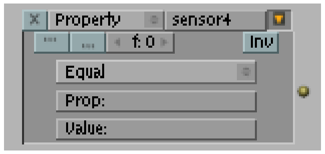
Ray - This will trigger when an object is detected along a certain axis. You can additionally check for the detected object having a certain material or property value.



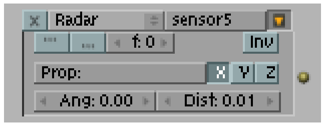
Random - Triggers randomly - change seed for different sequence numbers (or use python for real random generator).



Property - Triggers when a property changes, is between certain min and max values, or is equal or not equal to a certain value.

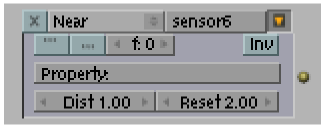


Radar - Triggers when an object is detected within a certain range (distance, and angle). You can specify a property that the detected object must have.

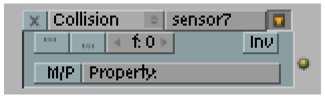


By Mal Duffin

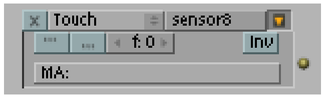
Near - Triggers when an object is detected within a certain distance. You can specify a property that the detected object must have.



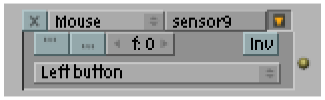
Collision - Triggers when the object is in collision with another object. You can specify a material or a property that the collided object must have.



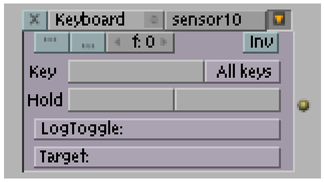
Touch - Triggers when an object is touching another object. You can specify a property that the touched object must have.



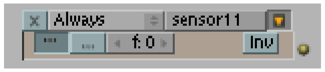
Mouse - Triggers when certain mouse events occur, such as mouse button clicks, mouse movement, etc.



Keyboard - Triggers when a certain key is pressed.

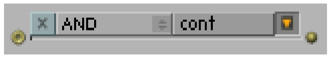


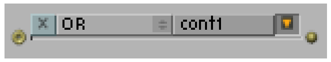
Always - Triggers every single frame.



Controllers: Controllers are triggered by their attached sensors.

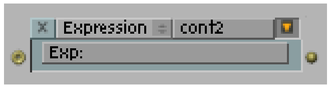
AND - Runs the connected actuator if ALL of the connecting sensors are triggered.



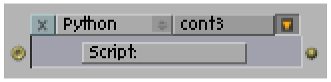


OR - Runs the connected actuator if ANY of the connecting sensors are triggered.

Expression - Evaluates an expression.

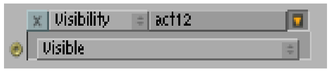


Python - Runs a python script.

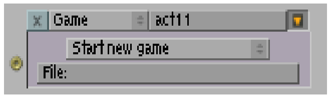


Actuators: If the relevant sensors are triggered, the controller will call the connected actuator(s).

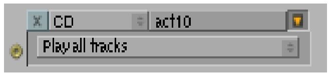
Visibility - Show and hide the current object.



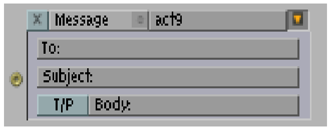
Game - Restart and Quit the current level. Can also load a new scene.



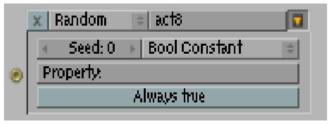
CD - Allows for control over CD music tracks.



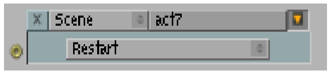
Message - Send a message to all objects, or to a certain object. This message will trigger the Message Sensor.



Random - Sets a random value into a property of the object.



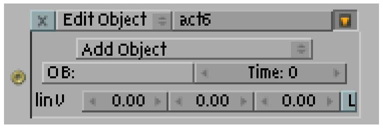
Scene - Allows for control over scenes - loading, playing, suspending, etc.



This is very useful for showing different scenes, such as a start-up scene or menu. When the user wants to play the actual game, a keyboard sensor (e.g. press [Space] to play) could be connected to a scene sensor, which would then load up the game scene.

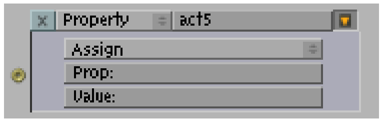
This actuator also allows you to specify what camera to look from, within a 3D scene.

Edit Object - Allows for control over adding, editing and deleting objects within the scene at run-time. This could be used to fire bullets from a weapon.

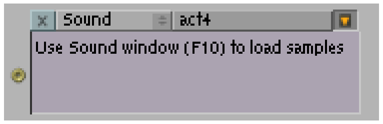

Edit Object act6
Add Object
OB: Time: 0
linV 0.00 0.00 0.00 L

It also has an object tracking feature.

Property - Sets the property value of the object (or of another object).

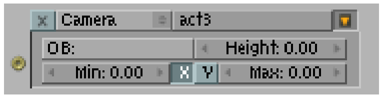

Property act5
Assign
Prop:
Value:

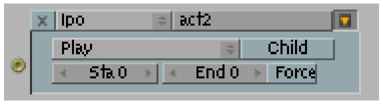
Sound - Allows you to control sounds from within Blender.

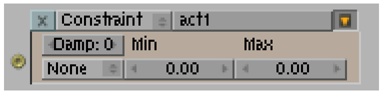

Sound act4
Use Sound window (F10) to load samples

Only sounds that have been loaded into Blender will be accessible.

Camera - Allows the camera to track an object. The camera can be placed behind the object (in either the X or Y axis) and forced to stay within a certain distance (Min and Max) and at a certain Height.


Camera act3
OB: Height: 0.00
Min: 0.00 X Y Max: 0.00

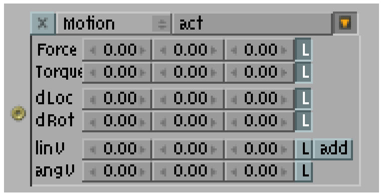

Ipo act2
Play Child
Sta 0 End 0 Force


Constraint act1
Damp: 0 Min Max
None 0.00 0.00

IPO - Allows control over playing object animations.

Constraint - Constrains the object's position.

Motion - Allows control over the motion of the object. This includes direct positioning and rotating of the object (dLoc and dRot), as well as applying forces to a physical object to move it (Force and Torque).


Motion act
Force 0.00 0.00 0.00 L
Torque 0.00 0.00 0.00 L
dLoc 0.00 0.00 0.00 L
dRot 0.00 0.00 0.00 L
linV 0.00 0.00 0.00 L add
angV 0.00 0.00 0.00 L

Continuing your Blender Game Engine

I hope you enjoyed learning the basic concepts of the GE. Hopefully you will continue to work with the GE, using the basic skills you have learned here, improving them with plenty of practice, and by learning from and sharing with members of the large Blender GE community.

Additional links and tutorials

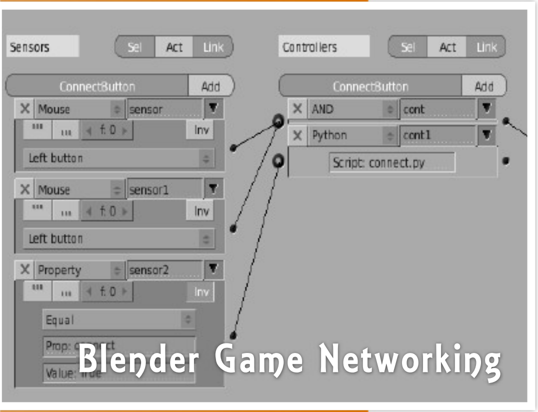
Blender Artists Community Forum - GE Section

This is one of the best resources for GE users.

If you want to ask any questions about how to do something with the GE, post some examples of your current game, or just keep up to date on all things GE, this is THE main place to visit. ●

http://blenderartists.org/forum/forumdisplay.php?f=34

By Mal Duffin



by Brian Cordell Hynds

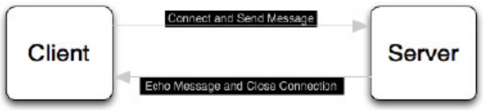
Introduction

Through this tutorial, we will create a working server/client for use in the Blender Game Engine. You will be introduced to low level Python networking through the use of sockets. Everything will be covered step-by-step and illustrated, and the completed tutorial files are available for download.

PART ! Networking And Scripting

Let's take a minute to explain a few things about networking. I could write an entire encyclopedia on computer networking, but I don't feel you need to know much more than the very basics of what a packet is. You don't need to fully understand this next part, but it's included for the sake of those who may. When a computer needs to send data to another computer, it sends a packet of information across a physical medium to the receiving end. This packet is encapsulated with header information that tells where the packet needs to go. When the packet is sent and received, it flows through a standard known as the OSI model. The only thing you need to know about the OSI model (and the TCP/IP model) is that it works. Packets are sent through a service that is associated to a port. A port is simply a number that TCP uses to send and receive information.

Now that I've summarized about six months of networking school into one paragraph, take a look at this next diagram:



This is a simple chart demonstrating how we will be setting up our server and client. The client will connect to the server and a socket connection will be established. The client will send a string of data to the server. The server will then read the data and send it right back to the client and close the connection. That's it. This model is known as an Echo server, since it's sole function is to repeat the data back to the client.

First, fire up Blender. To avoid the hassle of going step-by-step through creating a game menu, I've taken the liberty of creating one for demonstration purposes. You can download the file from the link at end of the article.

Lets take a look at what we have here. In a nutshell, there's a camera, 3 inputs (server IP, port, and message), along with a connection button. There's also a field for the server's response and an empty that we'll use for the user interface controller. Lets start hooking everything together.

To get some interaction with the game, we're going to need our cursor visible. We'll make a Python script that will do this. Open up the Text Editor window and make a new file called "onGameLoad.py". This will give me a good opportunity to introduce you to Python scripting, so the rest of the tutorial won't seem as intimidating.

With Python, we can add almost limitless functionality to our games. Logic bricks are a nice way to add simple functionality without knowing any code, but can become very complex when trying to add functionality that requires more then 20 brick connections. By using Python, we can avoid sorting through a web of brick connections, and extend beyond the limitations that logic bricks hold us to. However, without logic bricks, we would have no way to call our Python scripts.

Let's get back to the "onGameLoad.py" script. We want to make a script that will enable the cursor to be viewed in the game. The Python module that handles this is Raster-

By Brian Cordell Hynds

zer. To access the Rasterizer functions, we will first need to import the module into our script. After a module is imported, you will have access to all of it's functions. You can import the script using "import [module]". To save typing, we'll use the "as [name]" option. This helps tremendously when creating very long scripts that require you to type the module name a lot.

```
import Rasterizer as r
```

To show the mouse on screen, we'll use the showMouse() function like this:

```
r.showMouse(1)
```

Notice, to call the showMouse() function, we needed to reference the module first. To call any functions contained within a module, you reference them in "module.function().subfunction()" order.

Another thing to take note of is the function's attribute. In this case, I am using "1" to reference a 'True' value. 'False' is represented as "0". In fact, any non-zero value is a 'True' value. We could have also used "r.showMouse(True)" and it will work just the same. This is a matter of personal preference, but I think typing "1" and "0" is much faster and easier than 'True' and 'False'.

Now that our script is made, we need to create some logic bricks that will call them. We'll assign them to our empty, called "GameController". Select the empty and pull up the Logic Bricks window [F4]. Add an Always Sensor and disable True level pulsing. Then add a Python controller. This is where we enter the name of the script to be loaded. Connect the dots, and test out the game. You should now be able to see your cursor in the game view.

That should give you a good understanding of how Python scripts are handled in the game engine. Now let's get out of beginner mode and have a look at some real coding. From here on out, I'm not going to explain everything as "click this", "point here", etc. If you're seriously looking to add networking into your games, you should have a good understanding of the basics by now.

Part 2 : Setting Up the Python Server

Since we got our beginner course out of the way (quite possibly a primer for some), let's get our basic TCP Python server running. I like to use the Blender Text Editor to edit my scripts, mostly because I hate switching back and forth between applications. [CTRL+W] will save the .blend file, and [ALT+S] will save the text file outside of the .blend.

This next script will need to be saved outside of the .blend file, because it will be our server application. I have tried to find ways to run the server easily within the game itself, but I cannot find any easy way to do this. Running it outside of Blender will also make it much faster in responding and handling client connections.

So lets have a look at the heart of the networking relationship, the server. As I mentioned before, we're going to use Python sockets to make the connection between the client and the server. I'm going to show you the TCP Python server we'll be using, and within the code I'll break it down into understandable pieces using comments.

By Brian Cordell Hynds

```
# Let's import the socket module
import socket

# Assign a port number to a variable. Assigning values such as this makes
# it much easier to change, instead of having to edit every reference of the
# port in the script. Note, use a port value above 1024. Popular services, such
# as ftp, http, etc. use the lower port numbers and this will help to avoid con-
flict.
port = 2020

# Here is where we will create our socket. We assign the socket to 's'
by calling
# the socket module and socket() function. The socket() attributes tell
the socket
# to create a STREAM socket type from the INET socket family.
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Next, we bind the port to the hostname. This will tell the server that
we want
# to use the specified port to listen on.
s.bind((socket.gethostname() , port))

# Now lets tell the server to start listening for client connections.
# For testing purposes, we'll setup the server to handle 1 simultane-
ous
# connection. You can change this later if you need.
s.listen(1)

# Since this script will be run outside of the game, most likely in a
terminal
# window, we can use print() to show that the server is running. I've
included
# the port value in the command to display the port number.
print "Game server is running on port", port

# This is our main loop to handle incoming client connections.
# Connections are accepted and the message is stored into a varia-
ble.
# The connection information is print() 'ed to the terminal window
and the message
# is then sent back to the client. Then the connection is closed.
while 1:
conn, addr = s.accept()
data = conn.recv(1024)
# display client connection info
print "connected...", addr, data
conn.send(data)
conn.close()
```

Save this file as "server.py". If you created the file inside of blender, use ALT+S to save it outside of the .blend. Then open up a terminal window, navigate to the directory where you saved the server script, and type "python server.py" to run the server. You should see something like this:

```
$ python server.py
Game server is running on port 2020
```

To properly kill the process, use the "top" command and search for the "Python" listing under Command. Remember the PID next to it and press the Q key to exit top. Then use the command "kill [pid]" replacing [pid] with the PID of the Python command from top. After it's executed, you should notice a "Terminated" command at the end of your server.

By Brian Cordell Hynds

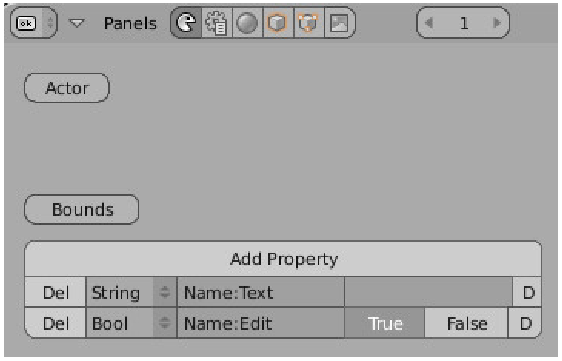
Part 3 : Creating the Client

Now that we have a running server, it's not going to do much if we don't create our clients. Using the .blend file from the previous articles (here), let's set a few things up before creating the client code.

First off, if you look at the menu, you'll notice there are 3 inputs: Server IP (input1), Port (input2) and Message (input3) are used to create the connection with the server. Currently, there's no easy way to enter the data for multiple text objects, so we'll have to script it.

Before we do, select each input object and create two properties. The first one being a string named "Text" and the second one as a bool named "edit". Set input1's edit property to True, and the others to false. This will give us somewhere to start.

I'm just going to summarize the script, instead of walking you through it step-by-step. We create a trigger that checks to see



Del	String	Name:Text		D
Del	Bool	Name:Edit	True False	D

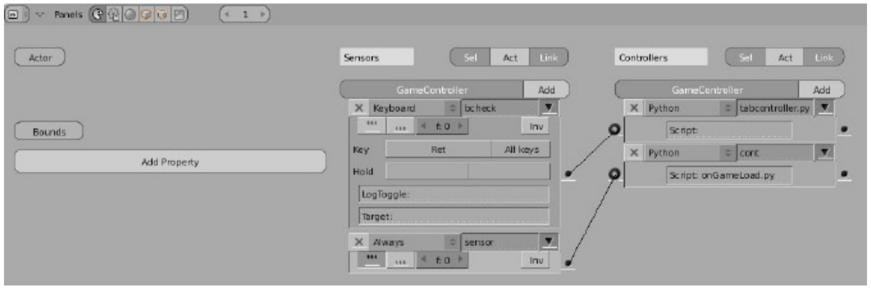
if a Sensor (a Keyboard Sensor) is positive. If it is, then we create a Python list of the 3 input objects. Then, we run an if/elif loop to set the "edit" properties of the appropriate inputs according to which conditional statement is true. Simple enough.

Here's the script. Save this as "tabController.py" in the Blender Text Editor.

```
import GameLogic as gl
cont = gl.getCurrentController()
trigger = cont.getSensor("tabcheck")
if trigger.isPositive():
# get objects
input1 = gl.getCurrentScene().getObjectList()["OBinput1"] # server ip
input2  =  gl.getCurrentScene().getObjectList()["OBinput2"]  #  port
number
input3 = gl.getCurrentScene().getObjectList()["OBinput3"] # message
proplist = [input1, input2, input3]
# server => port
if (proplist[0].edit == 1):
proplist[0].edit = 0
proplist[1].edit = 1
proplist[2].edit = 0
# port => message
elif (proplist[1].edit == 1):
proplist[0].edit = 0
proplist[1].edit = 0
proplist[2].edit = 1
# message => server
elif (proplist[2].edit == 1):
proplist[0].edit = 1
proplist[1].edit = 0
proplist[2].edit = 0
```

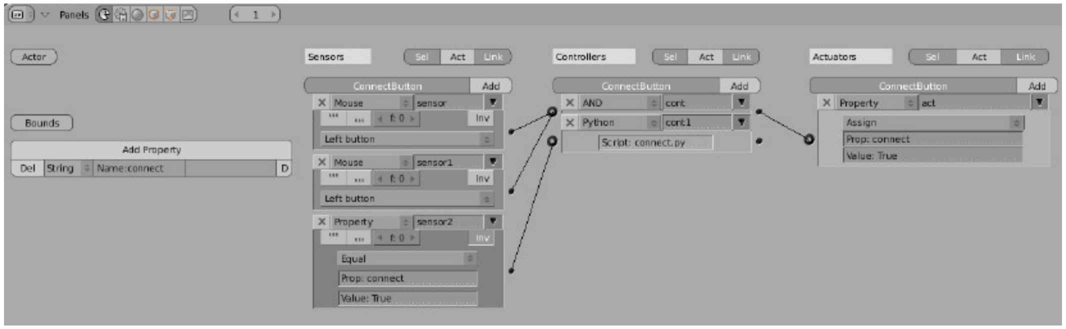
By Brian Cordell Hynds

Select the "GameController" empty and setup a Keyboard Sensor. For this exercise, I am going to be using the Return key, simply because using the Tab key actually produces "@" key returns which mess up the text input. You could use the Tab key if you want, but you'll have to edit the script to remove the "@" in the text object. Connect this Keyboard Sensor to a Python Controller, and use the "tabController.py" file.


Sensors: GameController — Keyboard: bcheck; Key: Ret; LogToggle:; Target:; Always: sensor
Controllers: GameController — Python: tabcontroller.py; Script:; Python: cont; Script: onGameLoad.py

Here's where a few problems lie within the logic brick design. What we need to do is create two Mouse Sensors ("Mouse over" and "Left button"), and when they are both positive a Python script is started. At first glance you might think, "Well, hook them both up to a Python script controller". This isn't going to work because each Mouse Sensor will act independently as if it were an OR condition, and the script will run every time a left mouse button is pressed and every time you mouse over the button. We need to connect them to an AND controller, but now we're presented with the problem of not being able to run the script directly. Since there is no Python Actuator, we're going to use the Property Actuator to set a bool property, which will trigger a Property Sensor that will initialize the script. Take a look at the image below to see how this works:

Onto the script. Much like the server script, I'm going to show you the script we'll be using, and then use comments within the script to explain what's going on.


Add Property: Del, String, Name:connect
Sensors: ConnectButton — Mouse: sensor, Left button; Mouse: sensor1, Left button; Property: sensor2, Equal, Prop: connect, Value: True
Controllers: ConnectButton — AND: cont; Python: cont1, Script: connect.py
Actuators: ConnectButton — Property: act, Assign, Prop: connect, Value: True

By Brian Cordell Hynds

```
# Since we'll be using Blender Game Engine functions,
# we need to import the GameLogic module.
import socket
import GameLogic as gl

# This is another problem with the logic bricks. We can initialize scripts using the
# Python controller, but we can't stop them when their sensor is not true. If we don't
# check the property, the script will actually run twice (on click and on release).
# We'll create a loop to test if the property is true, this way we only run it once.
conprop = gl.getCurrentController().getOwner()
if (conprop.connect == True):

# Now let's get the "Text" property values from the input objects, and assign
# them to respective variables. Notice that we need to use the int() function
# for the port settings. If we don't, the variable is assigned as a string, and
# you will get an error when binding the port because it's not an integer value.
hostinput = gl.getCurrentScene().getObjectList()["OBinput1"]
host = hostinput.Text
portinput = gl.getCurrentScene().getObjectList()["OBinput2"]
portstring = portinput.Text
port = int(portstring)

# As with the server, we create the socket.
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Instead of listening for connections, we'll connect to the server.
s.connect((host, port))

# We'll grab the message from input3 and use the the send() function
# to send it to the server.
message = gl.getCurrentScene().getObjectList()["OBinput3"]
sendme = message.Text
s.send(sendme)

# Assign the server response to a variable,
# so we can use it later, and close the connection.
data = s.recv(1024)
s.close()

# Now we can display the echoed message in the server response
# text object by assigning it to it's Text property.
resp = gl.getCurrentScene().getObjectList()["OBresponseValue"]
resp.Text = data

# And finally, we reset the connect property back to false.
conprop.connect = 0 Summary
```

Let's test it all out. Fire up the server, then play the game. Type in the IP of the server, the port number the server is running on, and a message you'd like to send. Press connect, and if all went well, the server should respond with your echoed message which is displayed in the response area. You can see the connections printed in the server terminal window. Hopefully, this will have given you a good understanding how simple socket connections are made and how you can get networking within your games.

You can download the entire .blend file, which includes the server and the game client, here.

In my next article, we'll cover how to design a simple Python chat server, complete with password authentication, and setting up the Blender UI to create a chat client in the game engine.

Until then, if you have any questions, comments, or problems, feel free to e-mail me at bchynds@mac.com, or jump into the discussion at BlenderArtists in this forum thread:

http://blenderartists.org/forum/showthread.php?t=111532 •

By Brian Cordell Hynds

MAKING OF: 'Orion Tear'

by Rogério Perdiz

Introduction

For those of you who are followers of the BlenderArt Magazine, I'm sure that at least one of the characters in the picture isn't a complete stranger to you. That's Orion, who was presented for the first time in issue #4 - Character Design. From that issue almost until the present day, he was working as one of the main actors of my 10 minute short animation, Orion Tear!

Orion Tear: Orion Tear is my first short animation movie. It was made with only Blender, Gimp, Inkscape, one workstation and me during 19 months and 7 days of non stop, from dawn till dusk, work. Although all the graphical part of the movie is done, the sound is still being developed by Telmo Cavaleiro, one local musician and sound EFX enthusiast, in his free time. It's release date is still undefined, but everything points to late Winter/early Spring 2008.

Origin: Ever since my eyes glazed watching the full motion videos of a video game called Final Fantasy 8, from the former company, SquareSoft, I knew that was the type of thing I wanted to do with my life. Considering that, at the time, I found the task of drawing a square with a ruler extremely hard... I also knew that I was in trouble!

Two years after that, I found an article in a Portuguese videogames magazine, describing the wonders of one revolutionary 3D application especially created for game development, called Blender 2.0. That precious 1Mb of pure creative power came in the CD, so I said to myself: "Well, I don't care about making games, but, what the hell; this is my chance to discover if I can do movies!"

And guess what? From that day on, I continually improved...until the day some guys made an open movie called Elephants Dream (ED) with the very same software I've

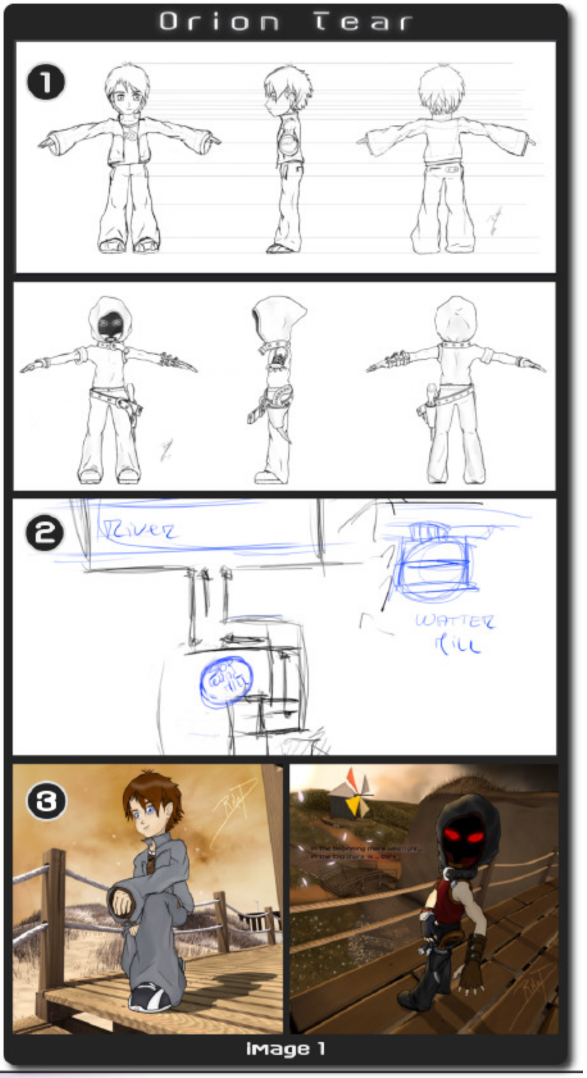


Image 1

by Rogério Perdiz

've learned to love so much.

At the time I still had alot to learn about animation movie making, but due to the open source spirit of ED, there was now a lot of learning material available... and also, like Aristotle said, "The things we have to learn before we can do them, we learn by doing them." So on the day 10-04-2006 the production of Orion Tear begun. Primary project purpose: Learn.

The article:

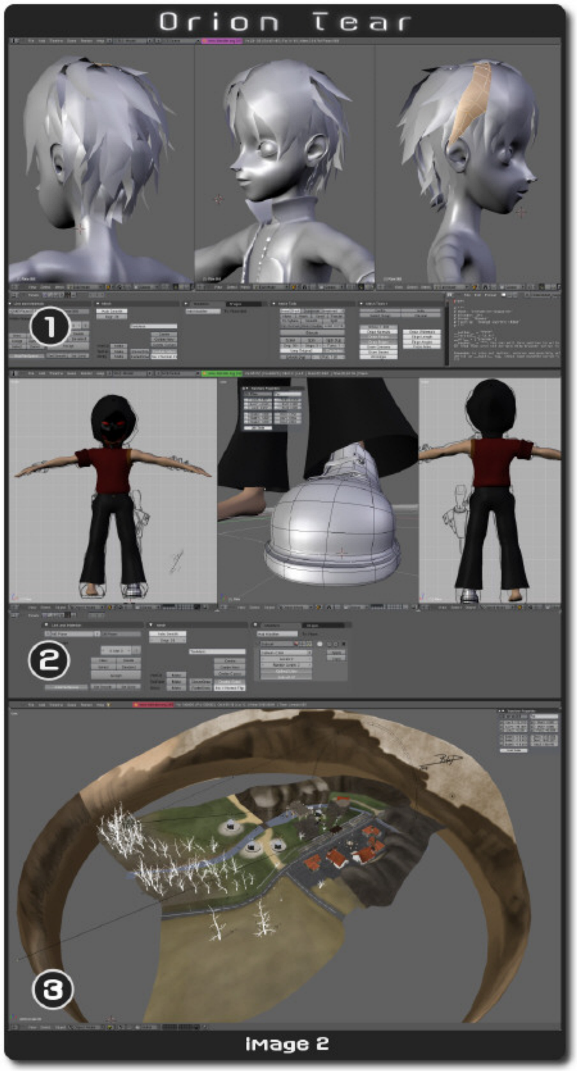
In this article I'll try to describe the steps that I've taken to accomplish the finalization of this Dante's feat, my difficulties, my thoughts, solutions and every thing else I can fit in, so stay tuned... if you can!

Sketches/Conceptual Drawings:

Although it doesn't seem that way, this was probably one of the most difficult tasks of the project or at least one of the most time consuming. Everything needs to be created from nothing.

For Orion (img.1-1) I had one basic idea. I wanted him to have the same body proportions of Zidane from Final Fantasy 9 and the look of my favorite character designer (Tetsuya Nomura) characters... mixed with my one drawing style.

For Dark, I didn't know what to do, but I knew what I didn't want to do... and that would be the classical Angel of Death look. Luckily at the time of creation, I saw one of Sergio Leone's western movies, and there you go, a dagger in a western pistol holder belt. Then I had, and still have, a cell phone that has 2 blinking, glowing red lights in the side and there you have the eyes. The belt in the neck was a mistake. I accidentally moved the layer of the belt up... and I thought to my self: "Hey! It kind of looks cool." The rest came from my natural character design style; I like gloves and unsymmetrical clothing.



by Rogério Perdiz

The sceneries simply evolved. I started to model them right away.

I wanted something that had kind of a natural feel with a subtle Fantasy look. Basically I've started from the look of one real beach place, near to my house, and then mixed it with some valley type of place. But something was missing, it simply wasn't catchy enough. So one weekend I grabbed my bicycle and went for a ride, without a fixed destination, hoping to find something to improve it. I still don't know how, but I lost myself and while trying to find my way back I found instead one very old wind mill. First I almost completely ignored it and kept on my way, still looking for something for my scenery but, after a while, I start to think:

"Darn!!! I haven't seen one of those mills working, for decades... funny to find one today... Hum! A mill... Mill!?"

I immediately squeezed the brake handles starting a sudden 180ć breaking stop, returned as fast as my bike allowed, took a bunch of pictures and after finding my way home I modeled it and there you have it.

After that I made some very rough sketches of the final scenery layout, having Mills as the theme (img.1-2) and then modeled a much too detailed version that turned out to be impossible to use... more of that in the next chapters.

After that I made some very rough sketches of the final scenery layout, having Mills as the theme (img.1-2) and then modeled a much too detailed version that turned out to be impossible to use... more of that in the next chapters.

Note: During this article, by scenery, I will be referring just to the main Mills scenery (img. 2-3). The movie has 3 sceneries, one in the mountains (img. 6-1) and another one that is a surprise!

Modeling Characters:

My model skills at the time were already pretty good, so, because I had defined all the details during the conceptual phase, I've just loaded the images to the blender viewports and start to model.

I started Orion with his Head, eyes and hair; then the body and finally the clothes. By the way, the Orion model that appears in the issue #4 had to be 80% remodeled due to problems in rigging, more on that in the rigging chapter.

For all the body and shoes I've used the point by point technique, meaning I've positioned all the points in space, based on the hand drawings, and only then joined them making faces.

The cloths were made initially by several nurbs circles that I've joined forming tubes (sadly I lost the screenshots of that and this is kind difficult to explain). For example: basically a shirt is only a subdivided tube that starts in one hand and ends in the other.

Then converting the nurbs tubes to meshes I could delete the faces that didn't matter. For example, the ones in the place where the body and neck passes through. After some adjusting and loop cutting what you get is a very good mesh topology shirt that deforms pretty good.

Scenery:

The scenery was my first problem. The first version was much too detailed, with lots of unnecessary polygons. I really didn't know about the limitations of my workstation, so I made tons of stuff until eventually everything moved excruciatingly slow. I thought: "well, it's slow but it should do the trick"! But it didn't. I had generated 2 millions polygons. Happy and full of joy with my self, I pressed Render and guess what? Right! 1 hour of render time which doesn't work very well for animation using only one PC, besides that Windows O.S. has an annoying 1.5 GB memory limitation and from some angles, rich in polygons, it didn't render at all.

by Rogério Perdiz

So full of sadness with my ignorance I removed all the stuff that didn't matter, meaning every thing that is never visible, like the bottom or the inside of the models, simplified them to the maximum and ended up with 150'000 polygons (img. 2-3) that rendered in 5 minutes.

But the funny part is that the high poly scenery wasn't a waste of time at all because I've rendered all the textures from it and applied them in the low poly models. More on that in the next chapter...

Texturing:

Well texturing was allot of fun! After I had modeled something I would, usually, texture it right away and only then pass to the next model and repeat the operation.

I didn't follow any specific method, besides my one method that still applies today. First thing I unwrap* the model and save the layout. Then I'll get out and take pictures of the real thing if it's easily available, things like concretes, woods, skies, cloths, etc. Then, normally, I'll first use them as reference and try to replicate them painting in Gimp or with blender procedurals.

For example: For the Orion Jacket (img. 3-1) I've first took photos of one real jacket, then in Gimp, based on the real jacket photos and in the UV Unwrap layout, I've painted all the stitches and jeans recognizable stuff, as also all the stone washed parts.

Although it already looked cool, it mainly looked what it was, painted. For it to look real I took one small square sample of the real jacket jeans texture and duplicated it a bunch of times until all the unwrap layout was covered, making sure at the same time, that it look seamless.

After that I've multiplied the painting on top of the texture and the final result is what you may see. All the clothes were textured this way.

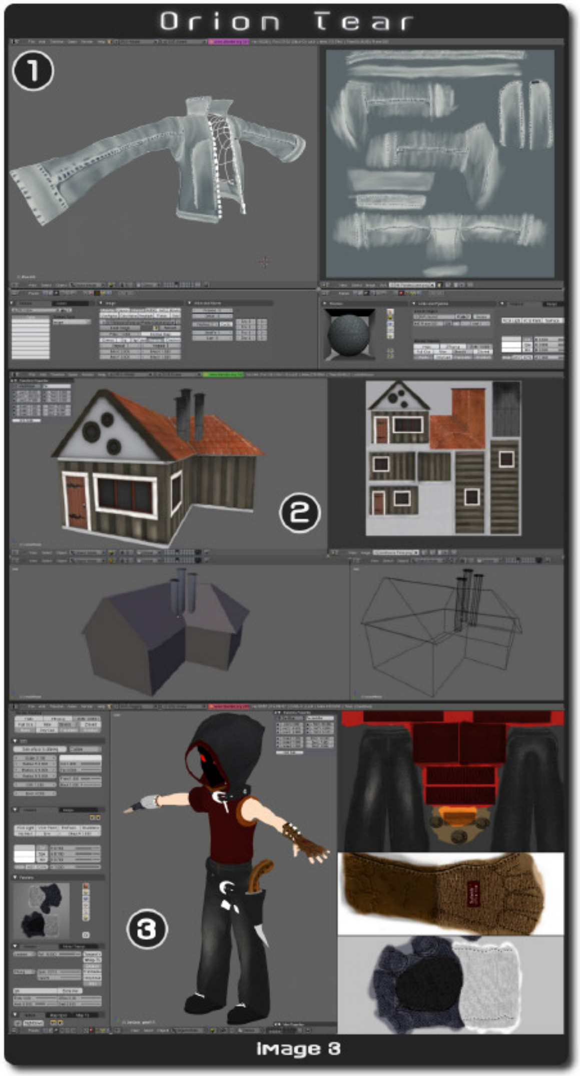


Image 3

by Rogério Perdiz

About the scenery:

Most of the textures of the low poly set are renderings of the high poly set enhanced in Gimp :) For ex.: the house (img. 3-2); from the high poly house I've rendered, only with Ambient Occlusion (still no baking at the time :P), one wall, then in Gimp I've added texture details, repeating the same procedure for all the walls, roof, etc., and them used it on the low poly model.

Almost everything was painted in Gimp, using after that the texture sample like I did for the clothes.

One thing that didn't quite work out has I expected were the far far away mountains (img. 2-3). They are made of one gigantic mate painting applied to a tube that surrounds the scenery. If you focus in them you will note that when the camera moves, a really bad, not wanted, distortion effect occurs. After some research I've learned that is a known issue in the world of 3DCGi, too big textures used that way seem to be propitious to cause that effect.

Lots of people ask me how I made the sky: Well the sky is just 3 blender procedural cloud textures mixed, and then the rest is a compositing trick. I also made allot of testing to try to have some movement on the clouds, but, it wasn't working at all so I moved on and left them static.

Most of the wood textures are procedurals based on the ones of the Blender materials Library, v 1.01 from Zsolt Stefan enhanced by me in Gimp and UV mapped to the objects.

***UV Unwrapping (from blender wiki):** During the UV unwrapping process, you tell Blender to map the faces of your object to a flat image in the UV/Image Editor window.

Rigging:

This was the troublesome one. When I started I didn't know anything about rigging, but thanks to Bassam Kurdali I was able to learn everything!

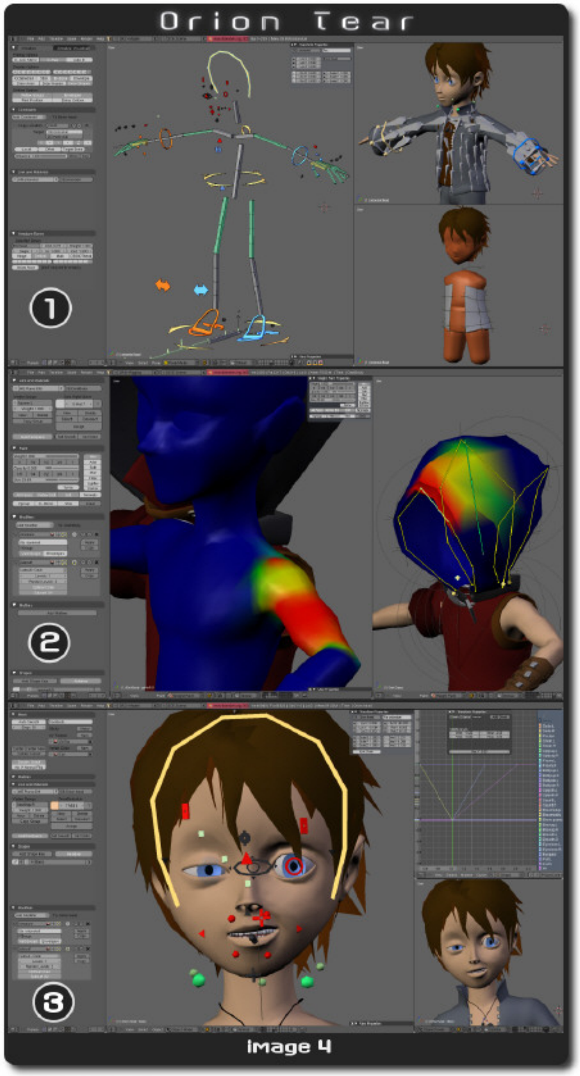


Image 4

by Rogério Perdiz

Bassam made one full rigged character called Mancandy and shared it with everyone. Considering that, at the time, it was the most complex rig and at same time easy to use I've ever seen I didn't had any doubt... But I didn't want to just import it to my characters; I wanted to learn, so I spent 2 weeks doing reverse engineering and creating the first Orion Rig. Well, after 2 weeks I had Orion rigged, but, I still didn't really Know how to make a rig, so every thing was a bit clunky. It was enough for walkcycles and basic movements, but I wanted them to be able to compete with Jackie Chan!

So I've started again another rig, this time for Dark, expanding my research to Emo and Proog rigs from ED. After another week disassembling bones, trying to figuring out all that, I suddenly started to see all the logic of the thing and progressively everything begun to make sense. Another 3 weeks passed until I successfully completed the Dark rig. After that I returned to Orion and because the first Orion mesh topology was pretty bad, for deforming, I had to almost remodel everything and remade the rig. After another month I had them completely ready for action.

I will describe now the rigging procedure used for Orion Tear:

- The first step is valid for all type of characters:
- We make the bone armature (img. 4-1);
- It's necessary to define witch vertex are influenced by each bone, we do that assigning vertex groups and using weight painting for fine tuning (img. 4-2);
- Corrective shape keys are added on top of everything to ensure proper deformation, mainly, on places like joins and pelvis.

The second step refers to the facial expressions:

- A reasonable number of expression shape keys are created;
- Control bones are added (img. 4-3);
- The shape keys are set to be driven by the control bones.

The third step refers to everything related to the dynamics like cloth and hair movement: For the cloths:

- I made a low poly version of the Jacket, that handles the softbody simulation
- I added Empties, one for each vertex of the softbody mesh and made them vertex parent of their correspondent vertex;
- Another armature constrained to the body armature was required. The tip of the bones of that armature coincides with the Empties location and has a 1chain IK constrain applied to them.
- A softbody deflector is required (the orange objects in img. 4-1) to avoid the cloth with body intersection. I made each deflector parent of the respective bone.

Control bones are added (img. 4-3);

Basically this will make the Empties follow the softbody movement, the bones follow the Empties and the cloth mesh follows the bones, originating an almost real time cloth simulation system.

The hair is a simple softbody with vertex painted influence control. That's why it isn't good, in fact I don't like the hair at all... not even the modeling, the texture nor the movement; I simply don't like it. But after spending so much time in it I didn't have the courage to remade it ... and now some people tell me that they love the hair, so hehe!

Animation:

Of all the other tasks of 3D animation movie making this one is my favorite.

I had made some animations on the past, but nothing with rigged characters. So I made the scenes by animation difficulty

by Rogério Perdiz

order... I started by the "easy ones" where the characters hardly move and ended up in the fight scene where you can see them jumping, fighting, rolling, falling, doing stunts etc.

Although the ones I thought to be the easy ones were in fact extremely hard. Just because someone doesn't move that doesn't mean they are frozen! I had to create tension movement, meaning the little muscle relaxation movements that we tend to make when we are stop. Also the wind in the cloths sleeves was key frame animated. The Orion jacket and the Dark Hood use a combination of dynamics simulation, with force fields deflectors, and hand animation. Most of the times I had to select the previously mentioned Empties and key frame them manually to correct their unrealistic movement.

The standard method used and still in use by me is:

- Make a little crappy storyboard with the movements for that particular scene;
- Go outside and film the reference moves;
- Import the reference film to the blender sequencer and make sure to have it in a little square preview window;
- Based on the reference video, set the basic poses;
- Work on the timing until everything looks reasonably natural;
- Fine tune the animation;
- Add the secondary movements like, sleeves, accessories, etc.
- Run the simulation for the Hair and Cloth dynamics;
- Correction of the dynamics by key animating the Empties;
- Small adjustments.

The fight scene:

The most difficult scene of this film it must have been the fight scene (img. 5-3). It has 2190 frames (1:28 minutes) of continuous action without cuts. This is where the Non Linear Animation (NLA) features of Blender came in handy.

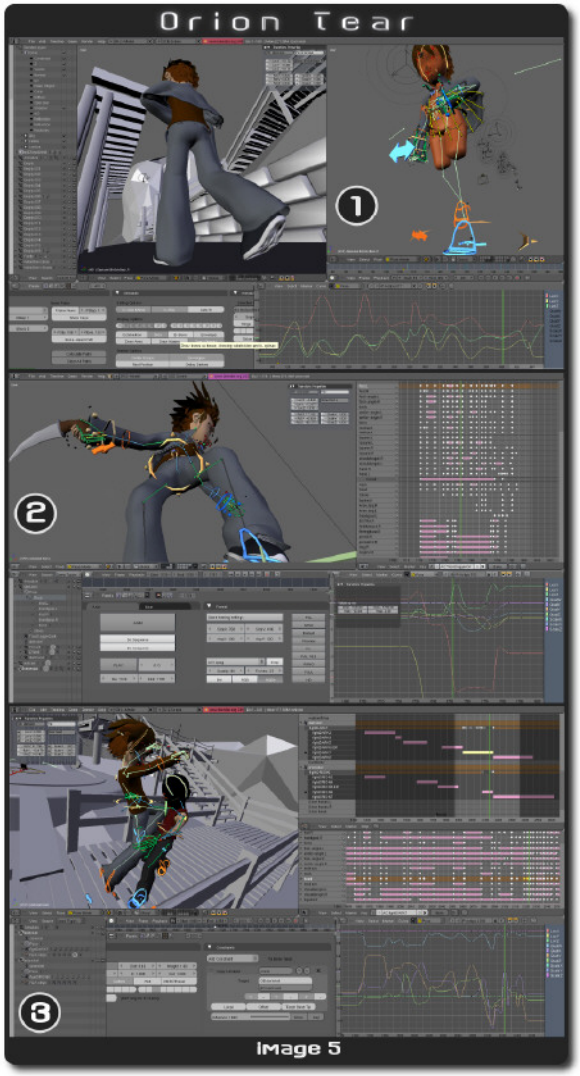


Image 5

by Rogério Perdiz

Thanks to it I was able to make 11 individual Actions sequences (6 for Dark and 5 for Orion) and Join them seamless in the NLA making it look a big one take Action. The camera was animated in only one continuous action, seamlessly completing the illusion.

It took me long time to do, and awfully longer time to render. This one scene uses all the tricks I've learned in all the others previous scenes; more on that in the next chapter...

Render/Compositing:

For me compositing is all about converting crappy renderings in stunning final images by whatever tricks you can come up with ;)

I really don't have any pre-established technique, this kind of thing I do almost by instinct I guess. Some people could call it luck... but that means I had luck 70 times (the number of scenes)!

The first thing needed is layers to mix. Because I couldn't render everything at the same time, nor really wanted to, most of the times I had 5 Base Layers to work with: scenery (to simplify the things a bit, the set, that is composed by more then 10 layers, renders just to one), shadows, ambient occlusion, characters and volumetric effects.

All the layers were rendered to OpenEXR, which allows better color control, and after composed the scenes were re-rendered to uncompressed Quicktime which allowed better hard drive usage and organization.

Lighting:

The scenery is rendered with Ambient Occlusion. It has very "few" lamps: three fill lamps at the sides; one Sun lamp that coincides with a Spot lamp for the sun volumetric halos; one lamp up and another down... plus a few ones along the river to try to simulate some Global Illumination (they are pretty much useless but I kept them, don't know why).

The characters use a Spot Lamp Dome* plus the sun lamp. The center of the dome coincides with the characters position and fol

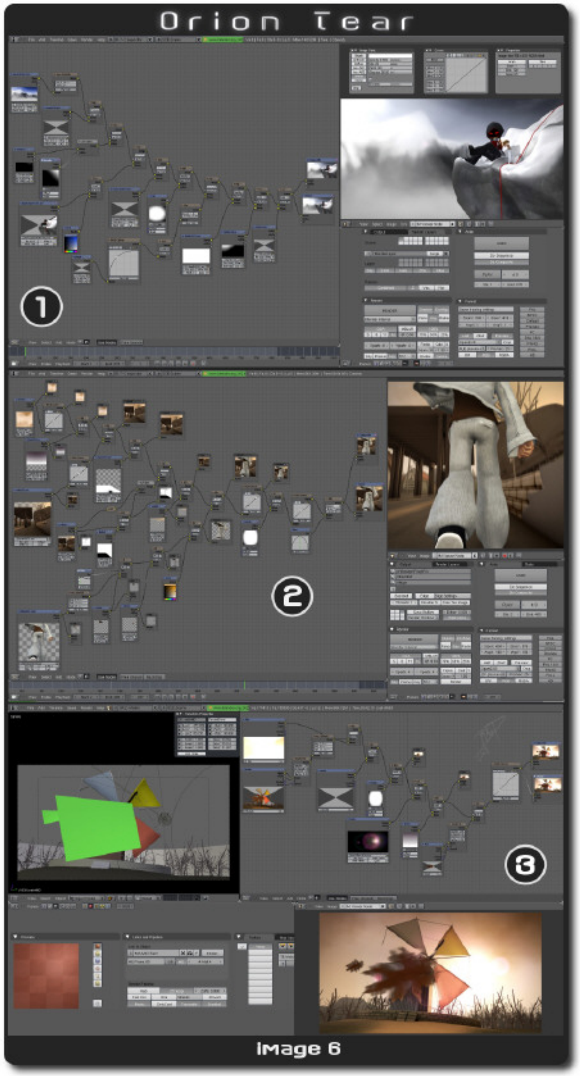


Image 6

by Rogério Perdiz

lows them wherever they go, resulting in a reasonable renderTime/quality global illumination look. No Ambient Occlusion is used for the characters. For some scenes I've use an elaborate and very time consuming technique for creating raytraced soft shadows (img. 6-2), that no one would note if I didn't mentioned it now and that I've abandoned in later scenes due the viability of the thing!

First I've rendered all the character with a shadow-buffer spot lamp dome, and then rendered the skin and the shirt with a raytraced shadows **spot lamp dome, but all in separated layers: Color, spec, shadows, etc, then reassembled them again in compositor but this time with the shadows blurred. Three or four scenes use this method!

By the way, I made two versions of the scenery, one for render and another, with much less detail for animation reference. The characters never set foot on the real scenery. The camera had to be imported from the characters file to the scenery file for all the scenes.

The biggest challenge was to be able to keep lighting and the mood consistent from scene to scene, because, most of it is just compositing work and needed to be done for every scene. The scenery had an, per frame, average render time of 5 minutes and the characters 7 minutes. Orion Tear took about 5 months to render, fortunately, most of them during the night. Everything was rendered with blender internal renderer.

* Refers to the top half of an Icosphere mesh that has a Spot Lamp parented to it and the DupliVerts plus Rot option turned on, resulting in having one spot lamp per vertex pointing inwards and towards the center of the dome.

** Raytraced shadows were very sharp edged. The current blender CVS version doesn't have that problem anymore.

Conclusion:

Well the primary purpose of this project was 100% accomplished. I've learned a lot... but I really don't want to make something like this again, alone I mean! ... and I don't recommend to anyone to try it.

This is just my, at writing date, opinion... If you also want to learn 3d animation movie making by making it, you probably shouldn't do more then 1 minute movies and you should focus yourself in only one particular task on each movie, but, who knows...

I could write hundreds of pages about Orion Tear, but I guess this article reasonably summarizes most of the things, so this is the end of the ride with me. Hope you like Orion Tear, for more news keep an eye on blenderartist.org!

Links:

Teaser: http://video.yahoo.com/video/play?vid=338577

Teaser-b: http://video.yahoo.com/video/play?vid=562159

Compositing: http://videos.sapo.pt/6ZsgTBaZLDZzAB0dIy4d

Rigging: http://videos.sapo.pt/2h21Mr9M5lEoZtzJGzjV

About my self:

I'm the guy that made Orion Tear, I've been using blender since version 2.0 and hope to be a better animation movie maker in the future ;)

e-mail:rogpertoons@yahoo.com •

by Rogério Perdiz

by Husam Ibrahim

Introduction

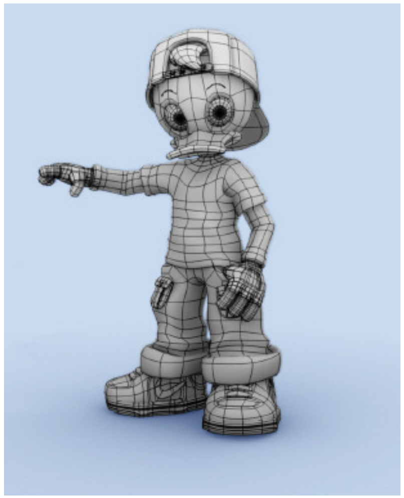
I've always been a big fan of Donald's nephews ever since I was a little kid. The new looks they received in Quack Pack and their portrayal as teenagers with distinct personalities made them cooler than ever. And now that I have laid my hands on Blender, I couldn't resist the idea of making one of them in 3D.

Modeling

I started looking for some reference pictures of the trio on the web. Louie was the one that resembled my view of what a teenager is like the most, so I decided to model him. I couldn't find perfectly matching front and side profiles and, to be honest, I was too lazy to draw them myself. So I decided to go with what I had. The head started out as a simple cube, subdivided twice and converted into a sphere using the To Sphere tool in the Edit Buttons (F9). Then I made the eye sockets by extruding a couple of faces inward. The beak was modeled using a poly-by-poly approach starting from a simple plane.

The T-shirt, pants and cap were all modeled by starting with circles, then extruding, moving and merging vertices as necessary. I then added some loopcuts [Ctrl+R] to the T-shirt and pants and moved the vertices to create the folds and wrinkles.

To model the hands, I started by making the fingers and worked my way backward from there. Then I made a duplicate of one hand, deleted the fingers and scaled it up a bit to make the glove. I had absolutely no idea how to make his shoes, so I looked for some reference photos. I found some nice looking trainers from Nike, so I modeled one of them and scaled it somewhat to give it a cartoony shape that fits the look of the character.



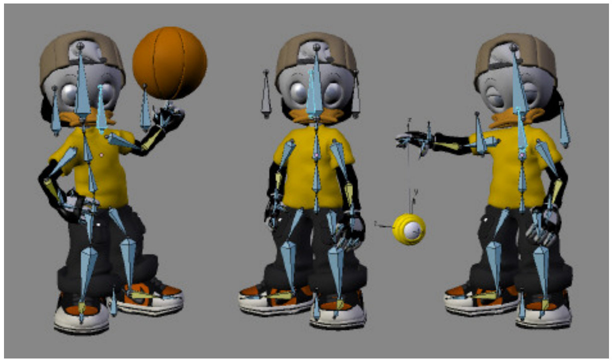
Rigging & Posing

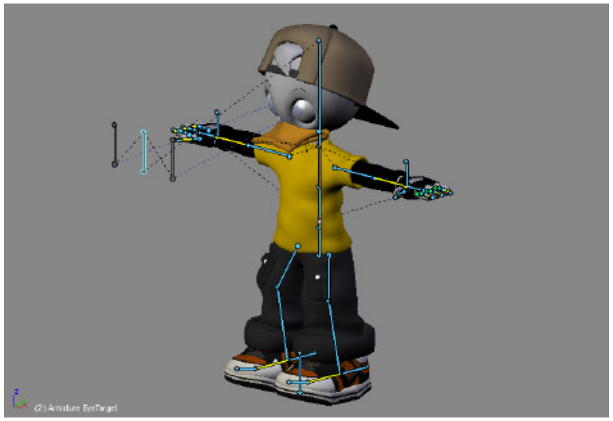
This is the part that I enjoy the most, despite my modest rigging skills, I must say. I went for a simple rig as I wasn't planning on animating this character at that time. The arms and legs were rigged using IK chains. The hands were rigged in the same way that was used in Emo & Proog's rigs. I also added two bones to function as Track To targets for the eyes.

he two bones were parented to a third bone that was used to control the eyes. Since I didn't have to cover a wide range of facial expressions, a couple of shape keys that control the position of the eyebrows were more than enough. I must admit this was one of my first complete rigs and it had many errors that nearly drove me nuts while posing the character.

by Husam Ibrahim

I guess I'll have to redo this rig when I start learning animation. I had planned since the beginning that I would put him in different poses against a white background. The poses had to be cool, show character, and have clear silhouettes that make the character instantly recognizable. I made seven different poses, two of which have never seen the light, as I didn't like how they turned out. Another two poses I was told didn't look good enough, so I settled for the three that you see in the final render.





Lighting

The lighting setup I used was fairly simple. It is illustrated in Fig.4:

1 Key light (Spot) from an upper right angle – Buffer shadows enabled.

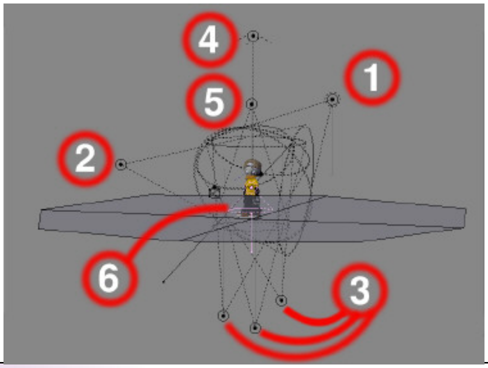
2 Fill light (Spot) from the left, a little bit lower than the key light.

3 Bounce lights (Spots). Those were used to simulate light rays bouncing from the ground on the character.

4 This Hemi light was used to light the ground. It had it's energy level finely tuned so that the degree of white on the ground matches the color of the background.

5 This light (Spot) was used to produce the orange spot on the ground. Of course, the ground was already fully lit by the Hemi light, so a normal orange light source wouldn't work. The only way to produce the orange spot was to get rid of the complementary color (blue). So this light source was negative with a blue color to subtract the blue and leave the orange.

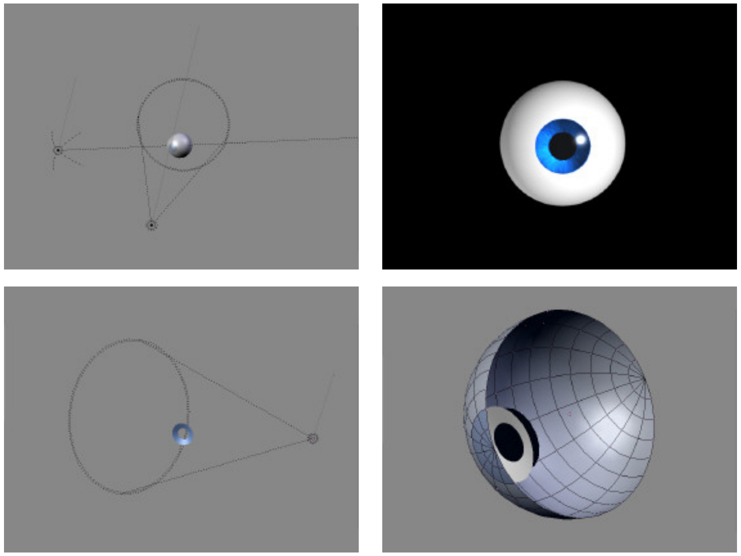
6 Shadow Only spot light to produce the shadow on the ground. This was important to show that the character's feet had a firm contact with the ground.



by Husam Ibrahim

The eyes had a lighting setup of their own. They had been modeled using the method described in the well-known "Creating Pixar-Looking Eyes in Blender" tutorial (Fig.5). If you like that method of making the eyes as much as I do, you must be using it and abusing it. But to get the most of it you really need to pay close attention to the lighting. Unfortunately, Blender doesn't allow assigning lights to different objects, so you'll just have to use different layers. I separated the irises from the eyes (to do this select them in Edit Mode, hit P, then choose "Selected") and then placed each eye and each iris on a separate layer.

Each eye was lit by a Hemi light and had a spot light carefully positioned to get a specular highlight on the cornea. Each iris was lit by a spotlight shining from an angle to get a nice color gradient across the iris. This really makes it look a lot better and gives some depth to it. Of course, you could achieve the same effect by using a Blend texture mapped to the Color channel of the iris with a Multiply blending mode, but controlling the light angle wouldn't be as easy. I finally parented the irises and the lights to the eyes so that they look the same wherever they point.



Materials & Shading

It was tempting at first to see what he would look like with 3D shading. I have to admit it was a lousy idea. Sometimes you just have to let the classics be the classics, if the term applies. So I decided to use a Toon shader for his body. What I don't like about the way people use the Toon shader in Blender is that they like to have those sharp transitions between lit and shaded areas.

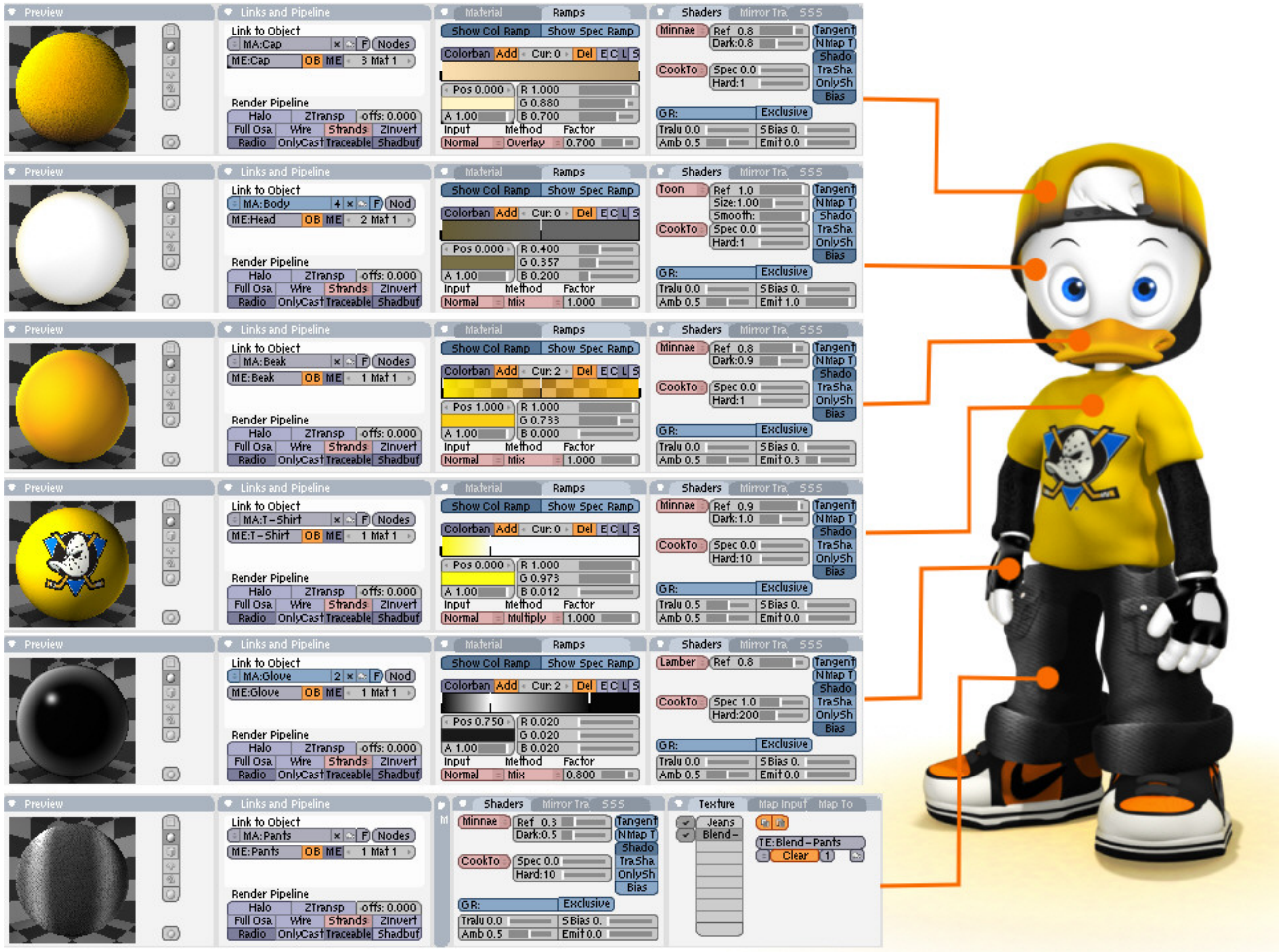
What people just don't seem to realize is that they can get nice looking toon characters by the exact opposite, a highly smooth transition between lit and shaded areas. It also helps if you give the shader a medium-to-high "Size" value to make the shaded area as small as possible, especially if your character is in front of a light background.

The other shaders I used were mainly Lambert for the glove and Minnaert for the clothes and the beak. It's really just a matter of playing with the settings to see which combination works best. Nearly all the materials I used had a ramp shader, which gives a nice looking, peach-like color falloff. For the body material, the ramp shader didn't only serve the aesthetics of the image, but it also helped separate the character from the background, as they had the same color.

The only textures I used were a jeans texture that I found on the web and the Mighty Ducks logo that you see on his chest. The logo needed some color adjustments so that it would match with the color of the T-shirt.

by Husam Ibrahim

Both textures were mapped using the basic texture mapping modes available in Blender, so no UV mapping was needed. The figure below shows the material settings I used.



```
MA:Cap     ME:Cap     3 Mat 1  | Show Col Ramp | Colorband Add Cur: 0 | Pos 0.000 | R 1.000 G 0.880 B 0.700 | A 1.00
           Input Normal  Method Overlay  Factor 0.700
           Shaders: Minnae Ref 0.8 Dark:0.8 | CookTo Spec 0.0 Hard:1 | Tralu 0.0 Amb 0.5 | SBias 0. Emit 0.0

MA:Body    ME:Head    2 Mat 1  | Show Col Ramp | Colorband Add Cur: 0 | Pos 0.000 | R 0.400 G 0.357 B 0.200 | A 1.00
           Input Normal  Method Mix  Factor 1.000
           Shaders: Toon Ref 1.0 Size:1.00 Smooth: | CookTo Spec 0.0 Hard:1 | Tralu 0.0 Amb 0.5 | SBias 0. Emit 1.0

MA:Beak    ME:Beak    1 Mat 1  | Show Col Ramp | Colorband Add Cur: 2 | Pos 1.000 | R 1.000 G 0.733 B 0.000 | A 1.00
           Input Normal  Method Mix  Factor 1.000
           Shaders: Minnae Ref 0.8 Dark:0.9 | CookTo Spec 0.0 Hard:1 | Tralu 0.0 Amb 0.5 | SBias 0. Emit 0.3

MA:T-Shirt ME:T-Shirt 1 Mat 1  | Show Col Ramp | Colorband Add Cur: 0 | Pos 0.000 | R 1.000 G 0.973 B 0.012 | A 1.00
           Input Normal  Method Multiply  Factor 1.000
           Shaders: Minnae Ref 0.9 Dark:1.0 | CookTo Spec 0.0 Hard:10 | Tralu 0.5 Amb 0.5 | SBias 0. Emit 0.0

MA:Glove   ME:Glove   1 Mat 1  | Show Col Ramp | Colorband Add Cur: 2 | Pos 0.750 | R 0.020 G 0.020 B 0.020 | A 1.00
           Input Normal  Method Mix  Factor 0.800
           Shaders: Lamber Ref 0.8 | CookTo Spec 1.0 Hard:200 | Tralu 0.0 Amb 0.5 | SBias 0. Emit 0.0

MA:Pants   ME:Pants   1 Mat 1
           Shaders: Minnae Ref 0.3 Dark:0.5 | CookTo Spec 0.0 Hard:10 | Tralu 0.0 Amb 0.5 | SBias 0. Emit 0.0
           Texture: Jeans, Blend-   TE:Blend-Pants
```

by Husam Ibrahim

Compositing

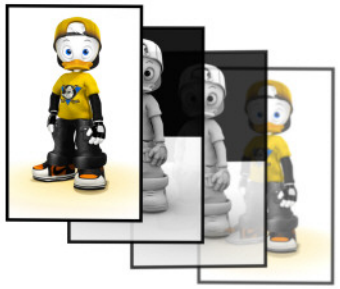
Not much was going on at this stage, except for an ambient occlusion pass I rendered to enhance the lighting of the image, and the motion blur effect for the yo-yo and the basketball. It's a shame that some potentially great renders fall flat just because they need a little bit of AO. Even if you are making a cartoony render, a little bit of AO always helps, unless you are planning on making your render super-stylized and 2D like. But just enabling AO and rendering doesn't help very much, as it comes out multiplied and results in a grainy render, even with a maximum number of samples.

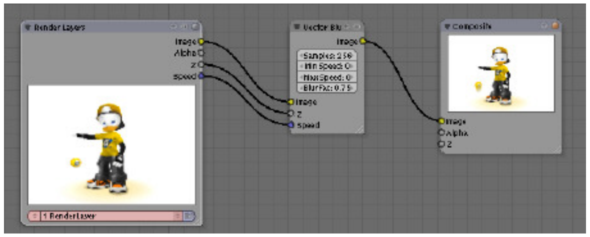
It's always better to render it as a separate pass and then combine it with your rendered image using the Node Editor, or your favorite 2D editing program. To render the pass, first enable "Ambient Occlusion" in the world settings, then go to your Scene settings (F10) and make sure "Ray" and "Do Composite" are enabled. Also make sure AO is enabled in the "Render Layers" tab. Finally go to the Node Editor, switch to "Composite Nodes", hit "Use Nodes" and connect the AO output of your render layer to the Composite node (Fig.10).

Once you have your AO pass, try combining it with your rendered image using different blending modes. For this image I used two layers of AO, one set to Overlay and the other one set to Burn. You should also blur the pass a bit and play with the opacity to get rid of the noise. Sometimes the AO pass can ruin your effort to have a special lighting for the eyes of your character, so you might want to erase that part of the pass.

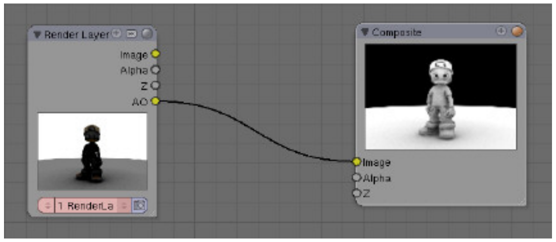
If you are using the composite nodes, you'll have to use the ID Mask node to do this job. There is a fair amount of information about this node in the Blender documentation. As a final touch I added the motion blur effect. To do this I used Blender's Vector Blur node (Fig.12), after animating the yo-yo and the basketball. Again, this is just a matter of playing around with the settings and the animation speed to see which combination works best.

Well, I guess this pretty much wraps it up. I hope this 'making of' proves somewhat helpful when the time comes and you decide to make a cartoony character.





Samples: 256
Min Speed: 0
MaxSpeed: 0
BlurFac: 0.75



Good luck and happy blending everyone!

Husam Ibrahim ●

MAKING OF: 'Monkey Game Project'

by John Buresh

Introduction

What is MGP? MGP (monkey game project) is an Open Source game project. Open Source means its free to download, play and alter it. So why did a bunch of people get together to spend all this time on a project they don't even get paid for? Well, thats a good question. MGP all started in the mind of one man.

MagicMyshu. Myshu was a member of a game developer forum. He gained an interest in 3D, but didn't want to spend $3,000 to get into 3D. Eventually, he discovered Blender. So he began his work in 3D, focusing mostly on modeling at the time. Myshu later joined a Blender forum called "Blender Artists" a place where blender users can display their work, and get tips or help on their project. After a few months of playing around with Blender, Myshu got an idea.

Myshu decided that he would make a small game project, in which people new to Blender could join and learn to make a game. Being a noob himself, he decided to call this project "Noobs UNITE!"

A bit crazy right? But it was a great idea! All Noobs were invited, even if they only had one post on the fourm.

"You're INVITED to join me! engine to work together and see what they can come up with."

"Warning: Game will be reasonably 'random' open mind is required."

We will not spend hours or days planning, we will just start to build our project and focus on the actual mechanical aspects for learning and experience sake."

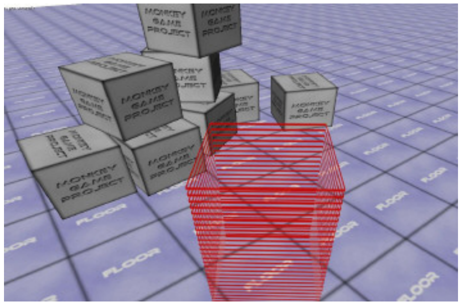
People showed interest in joining right away. Within 5 hours, a forum member called VenomSeven had modeled a level for the game. Forum members continued to show interest in this "Noob" project.

Many more people made small contributions. Mostly models, textures and sounds. Soon, Myshu decided to split the volunteers into teams, such as environments, programming, characters and animations and sound. The game progressed smoothly. The members of "Noobs UNITE" voted on a monkey game. The game then became known as Monkey Game.

All was going well, until the chaos of managing a project caught up to Myshu.

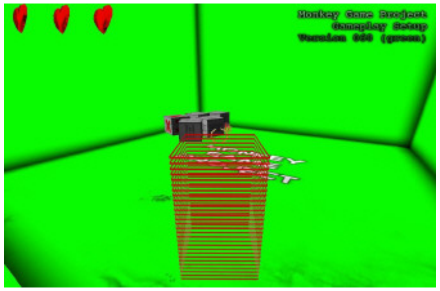
"I've sent/received probably about 500 emails/PM's about this project as well as spent 30 hours or so chatting on IRC about it – not to mention the 80 or so posts I've made on the forum about it too – while I was sick."

by John Buresh



Monkey Game began to crumble. Things went silent, people got angry; it was madness. A few members remained hard at work, determined not to see the project die. This was about the time when I joined. I knew very little about programming but decided to take a risk and jump into the programming team. The programming team had most of the things needed to make the game, we just had to make the game play.

Slowly, we inched on. I was in charge of health while another member, Stu_Flowers, worked on the inventory. The lead programmer, Chaser, worked on the game play set up. But something was wrong. We all knew it. Monkey Game was dead.



For quite some time Monkey Game lay on the hard drives of the few remaining members, collecting dust.

"No one has said it, but everyone's thinking it: the monkey game is dead."

But among all the bickering and yelling about the dead MGP, there was still hope:

"I have put way to much of my personal time and effort for the MG project to just die..."

And just like that MG (monkey game) was reborn, but this time, better, more organized, and more prepared for the realities of game creation. The former members of MG got back together and began to work on it once again. With new ways of communicating, there was much progress.

Within the next few months, the members decided to start over, and they began a new era of MGP. MGP now had a website that allowed the members to communicate with their community, since the threads on the forum were rarely updated. Over the summer of 2007, MGP members took a break, occasionally submitting new work.

In August of 2007, MGP was issued in the Blender Game Expose which was a great achievement for the members.

MGP was back on its feet, but the "leader", Myshu was nowhere to be found. We have not seen him since. This did not stop MGP.

So, How was MGP made?

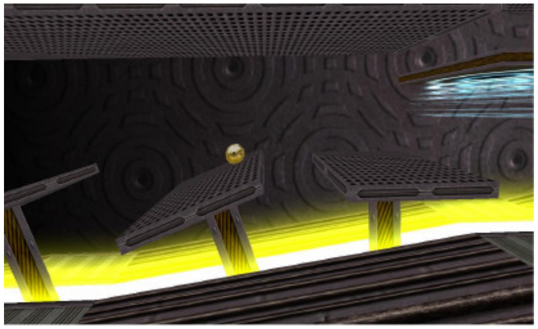
MGP was made by a bunch of people throwing pieces of work together to make a game. As I mentioned before, MGP was split into sections. There was Characters and Animations, which was lead by Myshu.

by John Buresh

They worked on all of the character modeling, including texturing, rigging and animating them. Another team was Programming. We worked on the game play setup and any logic and python needed within the game. The environment team worked on the entire environment that was used in the level; this included modeling levels and props, as well as texturing them. The Sound team worked on well, sounds. =) There was also a Mini-Game team, they worked on the mini-games that we planned to have in the game.

The reason why MGP didn't crumble apart is because all the teams worked seamlessly together. When the programming team needed new animations, the Character and Animations team was there with their blends waiting. Plus, we learned from our mistakes when it died the first time, communication was key. The programming team worked side by side, emailing each other whenever someone got a new idea or finished a script. The rest of the teams also worked together and made sure that each member was informed with the latest news.

Many days were put into MGP, almost a year passed since it started.



"We really didn't have a specific plan so we just threw together models, textures, layouts, and ideas and from this came a template we could use to make levels'

'We made a ton of content but 80% of it didn't even make it into the demo so we plan to release two separate versions of MGP."

As VenomSeven stated above, there will be two versions of the MGP demo available for download when we release, A regular demo version which will include a complete demo of MGP, the second will be a MGP demo Special Edition which will include the complete demo, and the files that did not make it into the demo, and possibly a few other things.

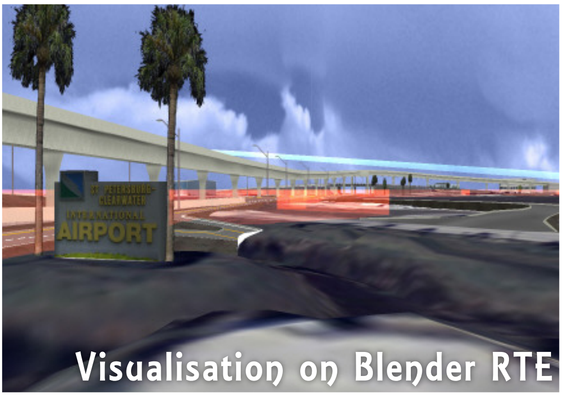
Programming?

The MGP demo will feature over 700 lines of code along with tons of logic bricks. The programming is based off of an empty system, which has empties that control most of the

features, parented onto the player. This gives the programming team an advantage when editing a feature, without messing up any of the others. Most of the python scripts are explained in the file, so you will just have to wait and see. =)

MGP has inched its way towards releasing a demo. There are very few of us left now, and with a release only a few days away, we hope we can show the community how much we have learned because that is what MGP is about. •

CASE STUDY: 'Of Blender's Real Time Engine on SR-686 Roosevelt Boulevard'


ST. PETERSBURG CLEARWATER INTERNATIONAL AIRPORT
Visualisation on Blender RTE

by Brian Treacy

Introduction

Blender's Real-Time Engine is not limited to just games. This article explains how I mixed Open-Source tools such as Blender and Gimp with proprietary tools such as MicroStation and Photoshop to produce a real-time walk-through of a proposed design of SR686 Roosevelt Boulevard for a presentation to Florida's Department of Transportation. Roosevelt Boulevard is located in Tampa, Florida adjacent to the St. Petersburg Clearwater International Airport.

As part of the final design plans to reconstruct a portion of Roosevelt Boulevard, my company (PB) conducted a study to add a major flyover ramp. The goal of the presentation was to show the client, FDOT, the relationship between the flyover and one of the glide paths for small aircraft at the airport. The software used consisted of MicroStation V8 and Inroads for the civil design work, Photoshop and Gimp for the aerial photography and textures, and Blender for the 3D modeling, animation, and real-time.

This is a realistic way to show how a proposed project fits within its context –an important element of the Context Sensitive Solutions approach to project development.

"CSS is a collaborative, interdisciplinary approach that involves all stakeholders to develop a transportation facility that fits its physical setting and preserves scenic, aesthetic, historic and environmental resources, while maintaining safety and mobility. CSS is an approach that considers the total context within which a transportation improvement project will exist. CSS principles include the employment of early, continuous and meaningful involvement of the public and all stakeholders throughout the project development process."

"What is CSS" FHWA 3 JAN 2008, 09:01:08 GMT.

‹ *http://www.fhwa.dot.gov/csd/what.cfm* ›.

Step 1 – Gathering the Existing Conditions

DTM stands for Digital Terrain Model; it is a triangulated surface mesh showing ground topology. Civil engineers use them as a platform from which to design roads and bridges. Professional surveyors generate them typically in the form of contours and spot-elevations. Engineers then import the data into CAD programs, i.e., GEOPAK, Inroads, or Civil 3D, and convert the data into triangles. Blender has no import scripts for the DGN file format, so I tried converting the 222,842 triangles to DXF but the file size inflated so large that Blender could not import them. Instead, I converted them to 3DS and imported into Blender.

At this point, I faced two additional problems: Blender's finite design-plane and 222,842 triangles. Having a finite design-plane, Blender arbitrarily moved my DTM off the proper coordinate system to a new location that Blender would accept. Having 222,842 triangles meant days of cleanup work removing extraneous detail to lower the total vertex count and then converting many of the remaining triangles to quads to make them easier to work with. To fix the first problem, I selected the entire contents of the DTM and from the southernmost part (an XY coordinate value of approximately 432000, 1295000) I moved it to a new coordinate value of XY= 0, 0. I could not fix the second problem.

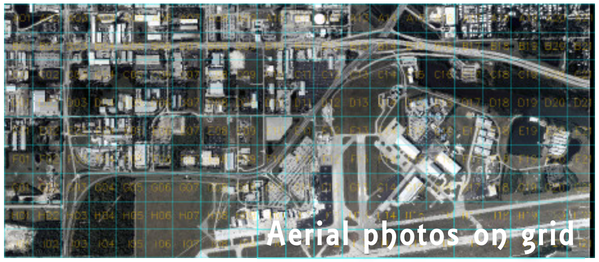
At the time (summer of 2005), Blender did not support Tangent Space Normal Maps, which would have moved the detail of the DTM into a texture. Therefore, I spent a few days manually cleaning up the surface mesh leaving

by Brian Treacy

the majority of the detail in the roadway curbs. The resulting mesh contained 171,606 polygons.

Civil engineers also use Digitally Rectified Orthographic Photos, which is a long term for scalable aerial photographs. Again, a professional surveyor flies over the corridor snapping pictures with a special camera then uses software (such as Descartes) to manipulate the aerials making them scalable. The aerials we received measured 10572 ft. by 4608 ft. at a resolution of 6 inches. Because of OpenGL limitations, I had to split the aerials into images measuring 512x512 px, which resulted in a 9x21 grid.

An engineer draped the grid over the DTM (Draping is a term similar to Blender's Retopology feature, where the vertices of the grid are projected to the DTM). This method makes the aerial photography align in the exact surveyed location on top of the DTM. I then used Blender's UV/Image


Aerial photos on grid

Editor to map the aerial squares to the draped image boundaries.

When dealing with a relatively flat area such as Tampa, a DTM is almost worthless for a presentation. There is not enough elevation-change providing visual-monuments—features at which people can look and recognize where they are. Therefore, you use buildings to provide this familiarity. Both survey and aerials provide the building footprints, but they do not provide any of the realism needed for the presentation. Therefore, we determined which buildings are visi-

Building UV map

ble from the road and took photos of them. The photos needed major cleanup, such as painting out trees, power lines, and cars—virtually any obstruction. I did this in Photoshop. I also extruded the building footprints in Blender to produce the walls followed by "eyeballing" the roofs. I used the UV/Image Editor to map the photos to the walls, and the roofs from the aerials to the roofs of the models.

To get a realistic skybox, I sent an engineer out to the Tampa Causeway on a beautiful day with a camera. She pulled over on the bridge and took a beautiful (almost complete without obstructions) 360 degree panoramic view of the sky. I stitched these together in Photoshop, then UV-Mapped them to a tube in Blender.

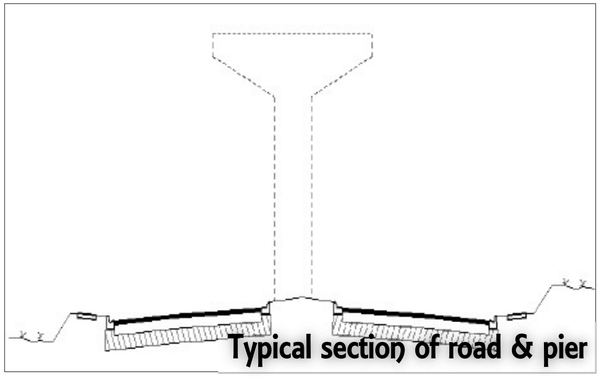
I scaled the top of the tube so that its radius was smaller than that of the base, producing somewhat of a cone shape. Finally, I created an IPO Animation of the sky with a slight rotation and assigned it to an "always" controller so the clouds would slowly move.

Stitched SkyBox

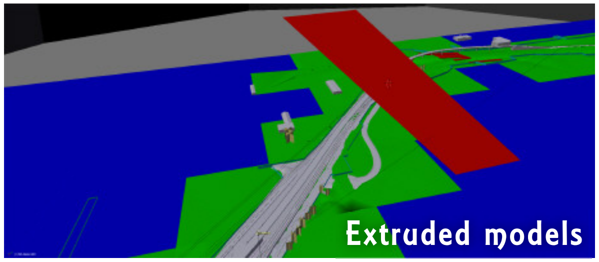
by Brian Treacy

Step 2 – Building the Proposed Design

To design a road, the engineer produces several types of elements: the horizontal alignment; the vertical alignment; typical sections; and cross sections. The horizontal alignment is a 2D curve depicting the pathway that the road, ramp, or bridge will follow. The vertical alignment (also known as a profile) is similar in function except it only provides the elevation differences. The typical section gives a general representation of what a cut-through of the road would look like, while a cross section gives an exact representation of a cut at an exact location. A standard project requires this for each road, ramp, and bridge.


Typical section of road & pier

I combined the horizontal and vertical alignments in MicroStation to produce one 3D alignment and imported it into Blender. Following the same workflow as used with the DTM, I moved the data in MicroStation first to address the coordinates and saved it to 3DS before importing. Once in Blender, I traced it using a 3D Bezier Curve. I then combined some of the typical sections and cross sections to form twenty-eight specific segments of six different roads, ramps, and bridges. I imported these into Blender and traced them as curves. Finally, I extruded the typical sections along the curves to produce the 3D models.


Extruded models

In roadway terminology, a gore is an area of transition where two or more roads converge or diverge. Due to elevation differences and adherence to design criteria, designing a gore in 3D is a challenge, especially when you want to make it usable in Blender's Real-Time engine. Essentially, I manually smoothed the meshes where the ramps joined the arterial roads and where the roads intersected. In retrospect, this was probably the most difficult task.

Designing the bridge for the flyover was very simple and similar to the roads and other ramps. I extruded the typical sections along the 3D curve to create the superstructure and manually modeled the hammerhead piers, approaches, and abutments within Blender.

Flyover & Piers

by Brian Treacy

Step 3 – Texturing and Painting the Design

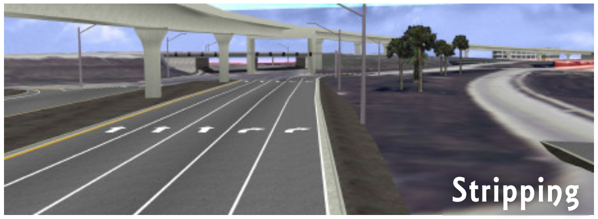
I collected some textures from photos of similar roadway projects. Textures I could not find, I had to create. I constantly toggled between Photoshop and Gimp to produce the textures; Photoshop for adjusting the colors and resizing and Gimp for tiling and transparency. I also did this while creating new textures from scratch using filters and plug-ins from both packages until I achieved the "perfect" blend. I UV-Mapped all of the models, then manually painted in the shadows with Vertex Painting.

Step 4 – Adding the Accessories

Accessories aid in the realism of the presentation. Therefore, I modeled traffic signals, street lamps, signs, trees, and the roadway striping. Based on photos taken within the vicinity of the project, I tried to model the accessories to look similar to what you would find in that area. For instance, the majority of the trees are palm trees, and the traffic signals are black.

Traffic Signals

Using a "billboard" technique for the trees, I added Alpha Masks to tree photos and UV-Mapped them to two crossing planes (forming an "X") in Blender. For the roadway striping, I used a 3D curve as a path and "dupliverts" to copy a white rectangular plane to represent the dashed stripes, and other 3D curves with a yellow or white plane extruded via "Bev Ob" to form the solid yellow or white lane lines.



I added a semi-transparent red "fence" representing the required right-of-way (property lines), and a semi-transparent blue plane representing the glide path to the adjacent runway.

Glidepath, 'Right of Way'

Step 5 – Adding User-Interaction

First, I would like to give a special thanks to Randall Rickert, for sharing his famous "walkthrough demo" – without his Python scripts and rig, this presentation would not have been possible. I imported Mr. Rickert's first-person viewer rig and scripts into my model. With so much distance to travel, the default settings took too long to travel from one end of the project to the other, so I made minor tweaks to

by Brian Treacy

some of the property values to increase the viewer's speed. Next I added collisions to specific faces (instead of to all of them) to keep the FPS up. Finally, I added a switch to toggle the visibility of the right-of-way fence, which gave the Client the ability to see his property limits when he wanted, but was able to turn them off when they got in the way of seeing other items of interest.

The Request for an Animation

After I saved out the executable and sent it with the DLL's to the Project Manager, he requested that I also produce a fly-through animation of about two minutes in length. This proved to be easier than I expected. I saved a copy of my presentation and added a path with a parented camera.

I also had to modify a few of the textures (mostly the ones with alpha masks), adding them as materials so that they would render correctly. After several low-resolution animations testing different flight paths, he selected his favorite, and I rendered out the final animation at exactly two minutes in length. UV Mapping and Vertex Painting cut the rendering time to mere minutes!

CONCLUDING THOUGHTS

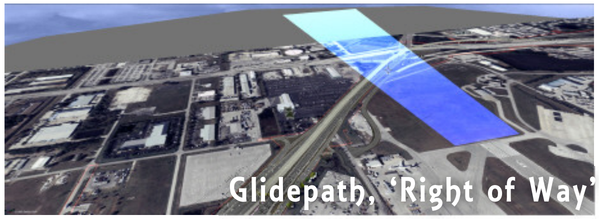
Accurately building and rendering a project during design (thus allowing our clients and the public to see the finished project while changes can still occur) aided the development based on CSS principles. Real-time presentation demands are growing in the construction industry, and Blender is adequately equipped to serve the demand.

Some helpful tools for Blender's future could include a MicroStation ".dgn" import/export script, an infinite coordinate plane (or even a floating coordinate plane), real-world units, and support of large high-resolution images in the Real-Time engine. Some helpful tools for my future include the use of Tangent Space Normal Maps and the Retopology tool to lower the vertex count and shift the detail from the DTM to the textures.

The entire presentation took approximately three hundred man-hours to create. The client loved it (the presentation), as did the Project Manager (though he did not love the three-hundred man-hours it took to complete), and Blender proved itself as a competitive tool in the commercial workplace. •

Brian Treacy

Senior Designer, PB



Interview with Mr. Blender! on Using Web Services in Blender

By - Hamed Zaghaghi

In this article you can learn how to use web services with Blender to make games or walk-throughs that are able to interact with resources on the Internet. I'll assume that you have some background of XML, Python, and PHP.

Bellow you can read my interview with Mr: Blender.

Me: Hello Mr: Blender!

Mr. Blender: Hey YOU!

Me: - I'm thinking about making a game or walk-through that interacts with others on the Internet.

Mr. Blender: OK, what's the problem? You can do it using python scripting.

Me: Yes I know that I can do it, but... I don't know how!

Mr. Blender: Do you know about web services?

Me: Web services?!

Mr. Blender: Yes. It's about exchanging data between a web server and a client.

Me: Hmmm. Can you explain more about web services?

Mr. Blender: Web services are methods you write or see in programs. Both are called with local programs, but the difference is that methods in local programs run locally and use local system resources, while methods in web services run on servers and use server resources. This isn't a complete definition of a web service, but it's close enough.

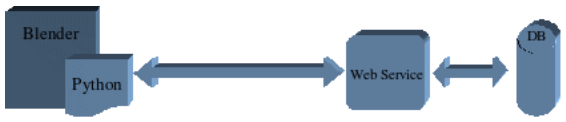
Me: Can you give me some examples of using a web service in Blender?

Mr. Blender: Okay. For example, you might want:

- Users of your game to register on your web site and then login into the game using their user name and password (you can see this in almost all online games)
- Your game to check for a newer version when starting and prompts the user to update
- To make a 3D presentation of a house for sale and users can chat with the house owner and ask him/her some questions, or many other examples.

Me: Oh, well, I think that when users must register their user name and password, I should have a database that contains user data, but what is the relation between Blender, Python, Web Service, and a Database?

Mr. Blender: Have a look at the following picture; I think that it will help you find the answer.



Blender! & Web Services

By - Hamed Zaghaghi

Me: OK, can you go step by step to make a complete example?

Mr. Blender: Yes. Let's make a game with version checker, step by step.

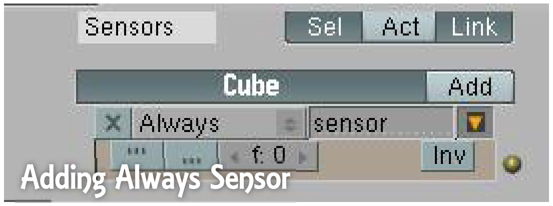
Me: OK. :)

Mr. Blender: First of all we need a game. You can use your previous games, or make a simple game, or simply use the standard Blender scene with a cube at the center as a game and continue.

Me: OK, I'll use a new Blender scene.

Mr. Blender: First we add an Always sensor, and uncheck both True and False pulse mode.

Me: Ok.



Adding Always Sensor

Mr. Blender: Now make a new file in the Text Editor and name it "check_version" without quotes, then add a Python controller and set Script parameter to "check_version" without quotes again, then connect the Always Sensor to the Python Controller.

Me: And then?

Mr. Blender: Before doing anything else in Blender, we need to configure our web service server. I use Apache as my web server, PHP for writing web services, the XML-RPC protocol for my web service, and Keith Devens' implementation of XML-RPC on PHP. You can find XML-RPC implementations in other programming languages at http://www.xmlrpc.com/directory/1568/implementations

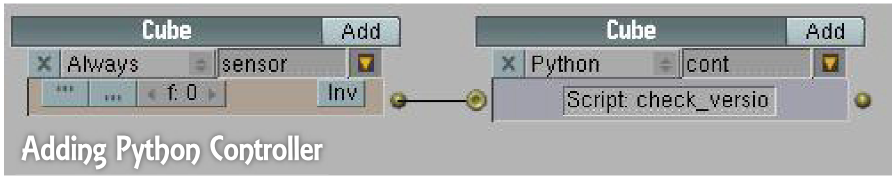
Me: Oh, and then?

Mr. Blender: For setting the web service up, first download Keith Devens' implementation from http://www.keithdevens.com/software/xmlrpc/source, then upload it somewhere to your web server.

Me: OK, I'm downloading it and uploading to http://en.blender.ir/webservices.

Mr. Blender: Then write this code in a text editor and save it as a php file in the same directory that you uploaded the XML-RPC implementation. Let's call it server.php.

```
<?php
#1)Include Keith Devens implementation
include("kd_xmlrpc.php");
#2)Web Service Methods
$xmlrpc_methods = array();
$xmlrpc_methods['CurrentVersion'] = 'getCurrentVersion';
$xmlrpc_methods['method_not_found'] =
'XMLRPC_method_not_found';
```



Adding Python Controller

By - Hamed Zaghaghi

```
#3)Web Service Server
$xmlrpc_request = XMLRPC_parse($HTTP_RAW_POST_DATA);
$methodName =
XMLRPC_getMethodName($xmlrpc_request);
$params = XMLRPC_getParams($xmlrpc_request);

if(!isset($xmlrpc_methods[$methodName])){
    $xmlrpc_methods['method_not_found']($methodName);
}else{
    $xmlrpc_methods[$methodName]($params);
}
#4)Web Service Methods Implementation
function getCurrentVersion($params){
    $returnversion['version'] = 3.60;
    $returnversion['minor']=60;
    $returnversion['major']=3;
    XMLRPC_response(XMLRPC_prepare($returnversion));
}
function XMLRPC_method_not_found($methodName){
      XMLRPC_error("2", "The method you requested,
'$methodName', was not found.");
}
?>
```

Me: I can do it, but I don't know what the code is for.

Mr. Blender: This code setups an XML-RPC server using Keith Devens' implementation, with two methods, "getCurrentVersion" and "method_not_found." The first method is the one we want to call from blender, and the second one is for requested methods that this server doesn't support. In the first section of code we include the XML_RPC implementation, and in the second section we setup an array that is used with the server code to find an implementation of the requested method. For example when in line 2 section 2, we say that if a method named "CurrentVersion" exists, the requested server should run the "getCurrentVersion" method implemented in section 4.

Then, in the third section, parse the request and extract the requested method name and its parameters. Finally, check if the requested method name was implemented, and if not, run "method_not_found" instead.

Me: Oh. I think that after we write this code and save it on server.php, we can then send a request method name and its parameters to server.php, and this code will run the requested method and respond with its value.

Mr. Blender: Yes, that's right. As a short explanation, the "getCurrentVersion" method responds with a structure of three fields - version, minor and major. Every time my program gets updated we change those numbers. In these methods, we can connect to databases and get some information from them.

Me: And then from python send a request to server.php to run getCurrentVersion. But how can we do it in python?

Mr. Blender: This section is easier than the previous one. Python has a library named "xmlrpclib," and we can use it to send requests and receive responses. To see what this library can do, paste this code in the python text editor of Blender, then press P.

```
import xmlrpclib
this_version = 1.3
server =
xmlrpclib.Server("http://en.blender.ir/webservices/server.ph
p")
```

```
current_version = server.getCurrentVersion()
if(current_version['version'] > this_version):
    print "Go to program web site and download updated version."
else:
    print "program is updated."
```

Me: I'll do it, but where I can see the results?

Mr. Blender: You can see the printed line in the terminal/command prompt of blender.

Me: Thank you, Mr. Blender, for your time.

Mr. Blender: You're welcome!

Interviewer - Hamed Zaghaghi,

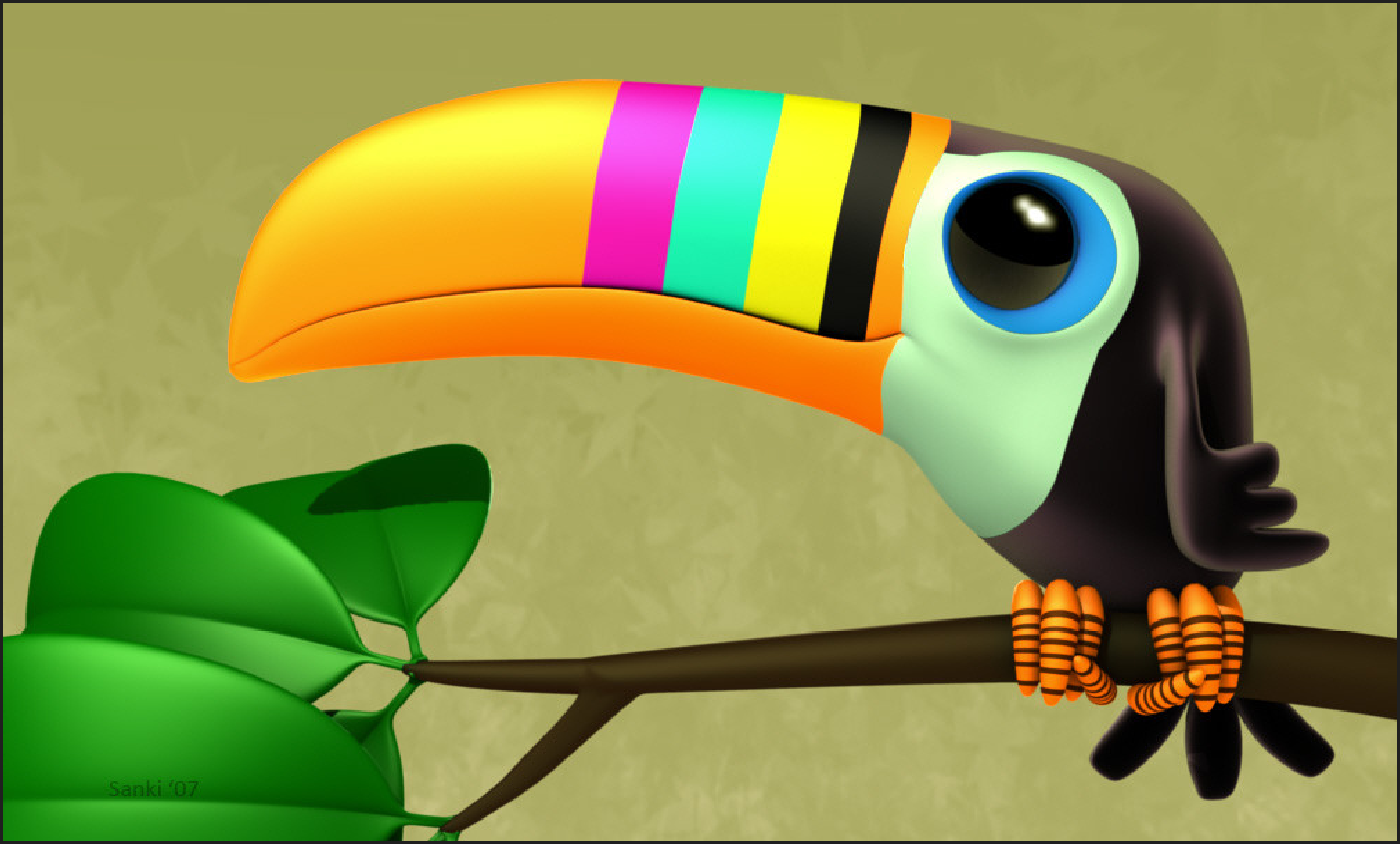
MSc. Student of Algorithms and Computation,

Engineering Science Department, University of Tehran

Email - hamed.zaghaghi@gmail.com •

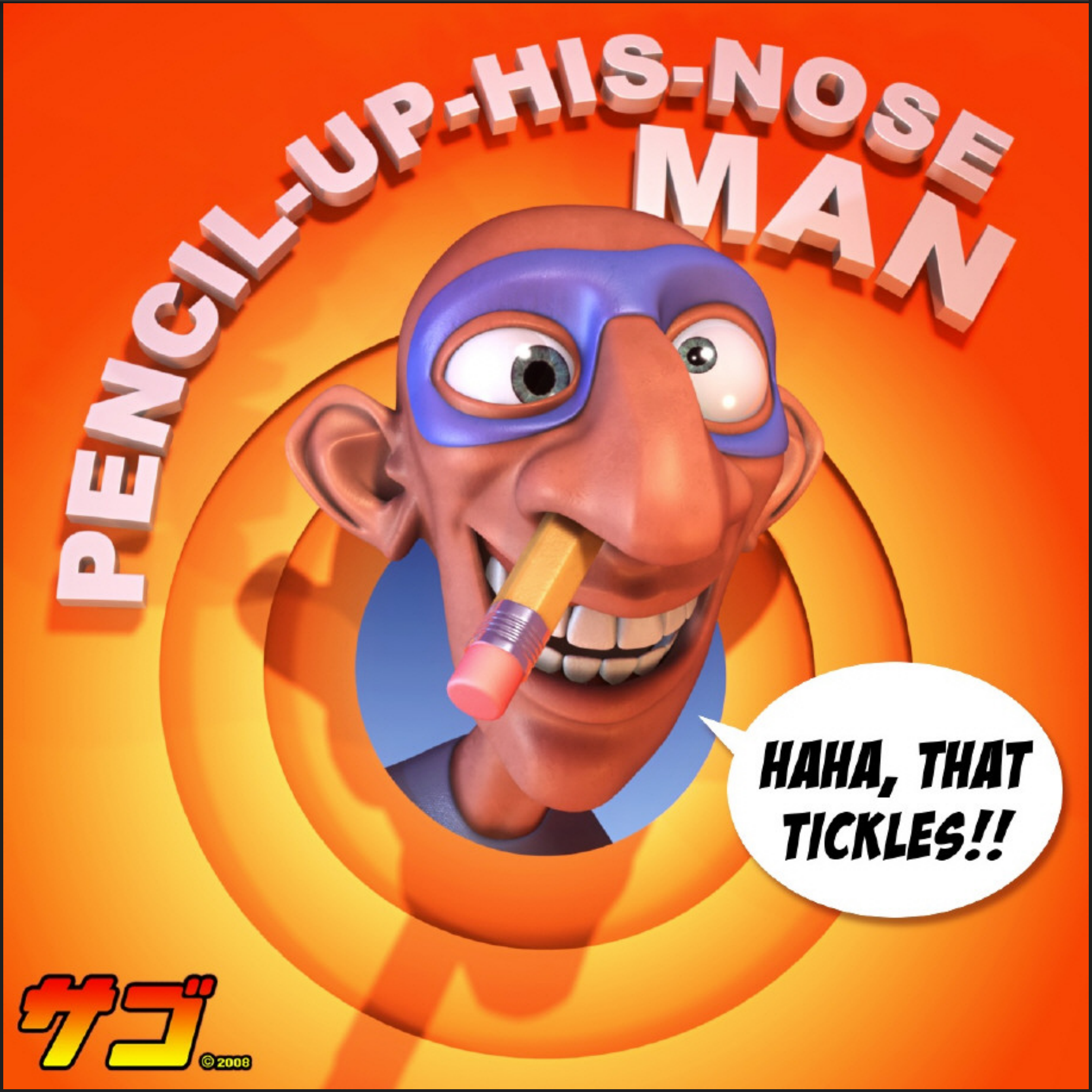
By - Hamed Zaghaghi

Maciej Sankowski (sanki) - CMYK Toucan

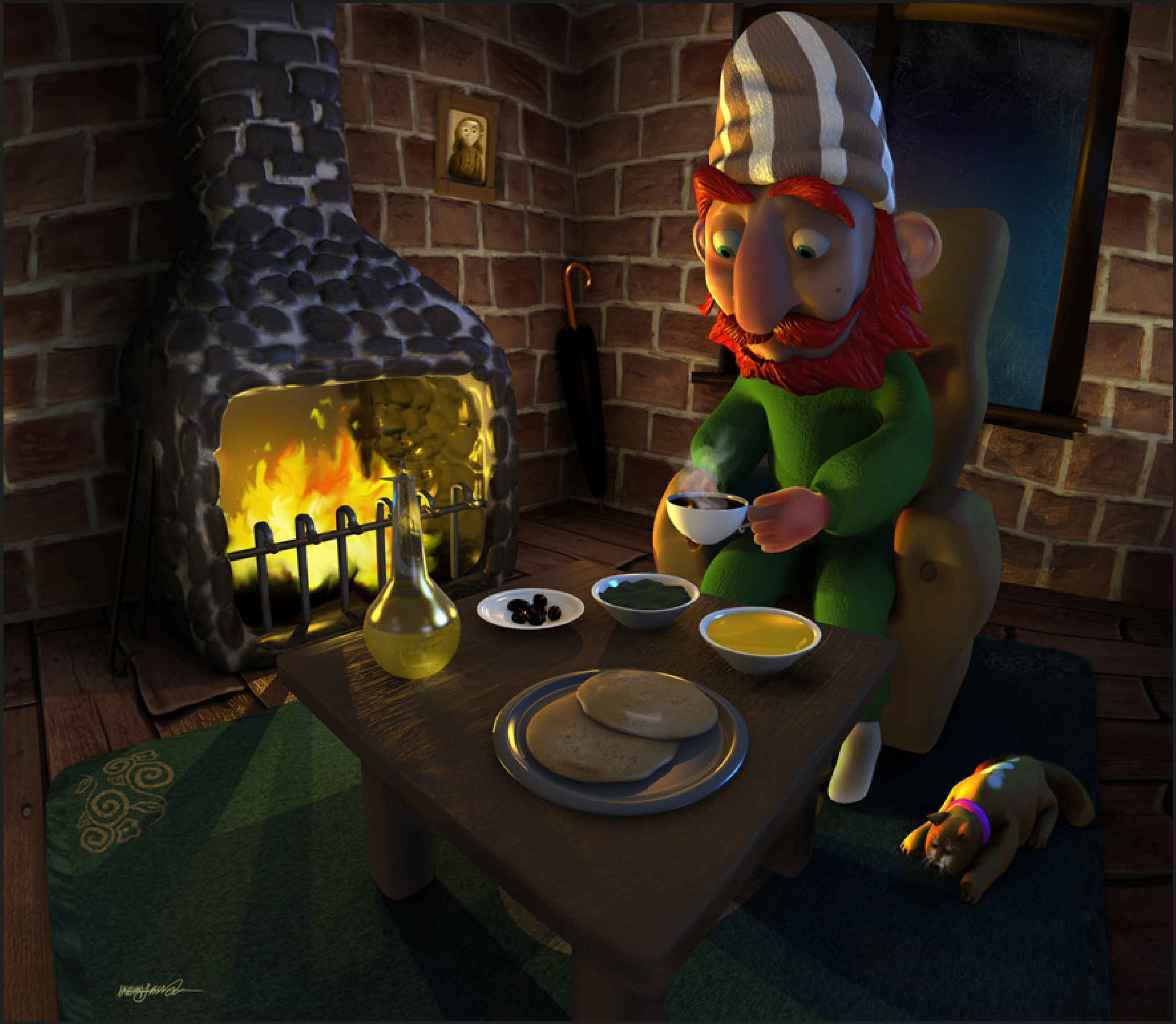


Salvador Gracia Bernal - Sad Love

Sacha Goedegebure - Pencil-up-his-nose-man



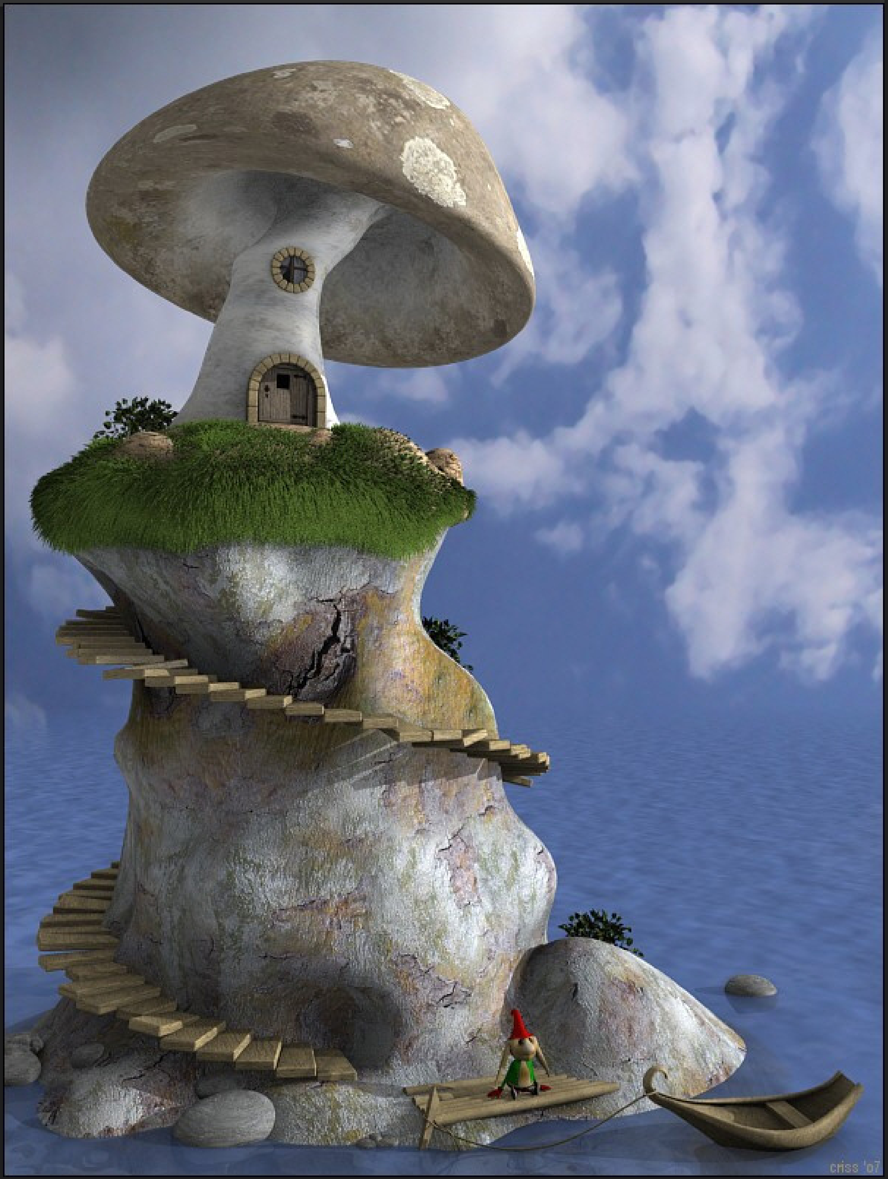
Al-Haitham Jassar - Winter



Cristian Mihaescu - Night of the dead

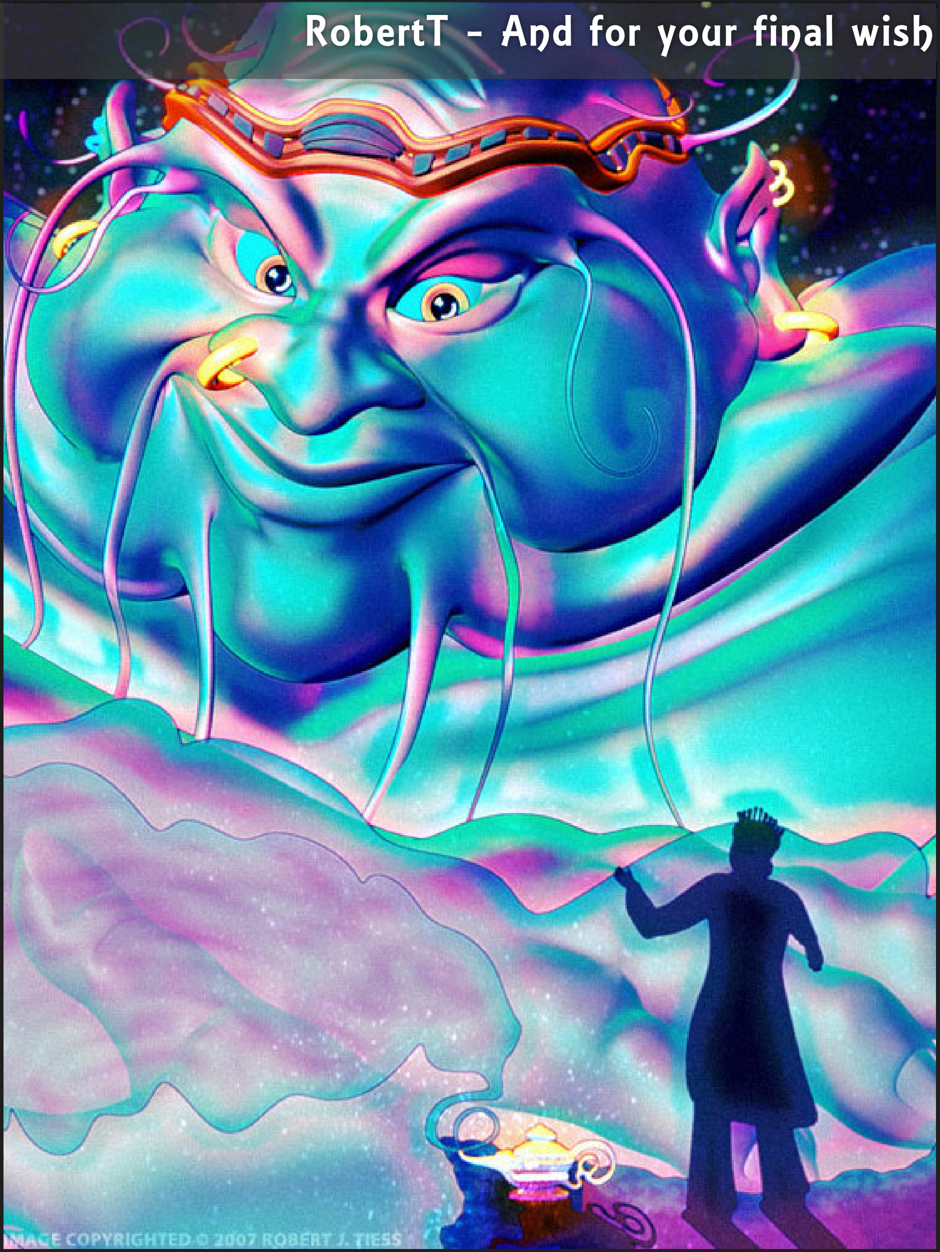
Cristian Mihaescu
criss '07

Cristian Mihaescu - Owner

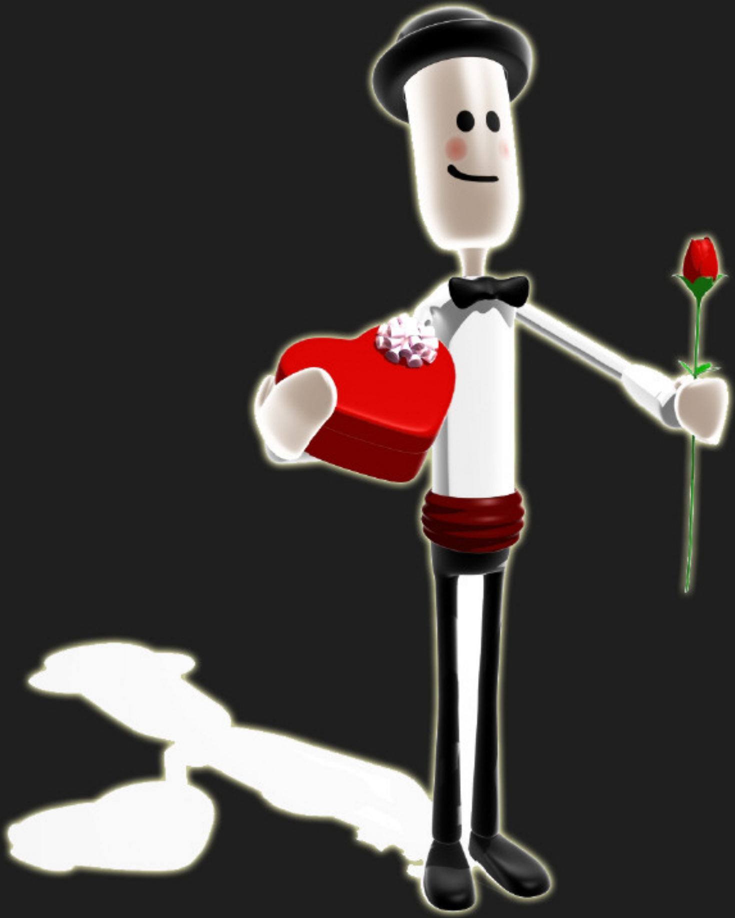


Adr Kiritschenko - Eet

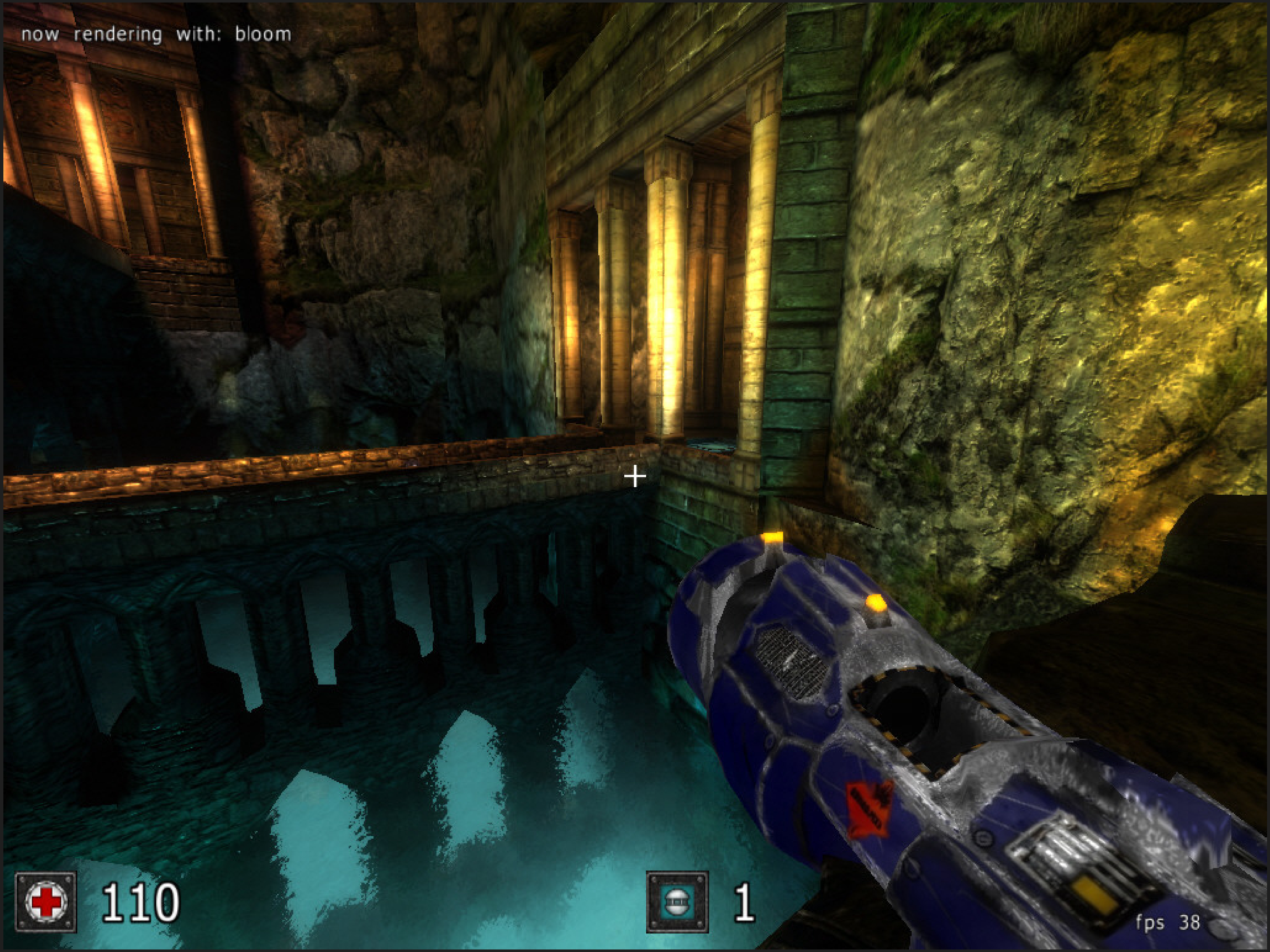
RobertT - And for your final wish



Bruce Parrott - A Gift



John Siar (geartrooper) - Grenade Launcher



John Siar (geartrooper) - Ironsnout

Want to write for BlenderArt Magazine?

Here is how!

1. We accept the following:

- Tutorials explaining new Blender features, 3dconcepts, techniques or articles based on current theme of the magazine.
- Reports on useful Blender events throughout the world.
- Cartoons related to blender world.

2. Send submissions to sandra@blenderart.org. Send us a notification on what you want to write and we can follow up from there. (Some guidelines you must follow)

- Images are preferred in PNG but good quality JPG can also do. Images should be separate from the text document.
- Make sure that screenshots are clear and readable and the renders should be at least 800px, but not more than 1600px at maximum.
- Sequential naming of images like, image 001.png... etc.
- Text should be in either ODT, DOC, TXT or HTML.
- Archive them using 7zip or RAR or less preferably zip.

3. Please include the following in your email:

- Name: This can be your full name or blenderartist avtar.
- Photograph: As PNG and maximum width of 256Px. (Only if submitting the article for the first time)
- About yourself: Max 25 words .
- Website: (optional)

Note: All the approved submissions can be placed in the final issue or subsequent issue if deemed fit. All submissions will be cropped/modified if necessary. For more details see the blenderart website.

Upcoming Issue 'Theme'

Issue 15

Animation

- Recent improvements in animation tools
- Facial animation: emotions, lip syncing, phenomes
- Making you animation more believable: arcs, follow through, secondary actions
- Free characters and character rigs
- etc.

Disclaimer